Muhammad Nadeem

Adaptive, Low-power Architectures for Embedded Multimedia Systems

with focus on H.264/AVC video codec

Adaptive, Low-power Architectures for Embedded Multimedia Systems

with focus on H.264/AVC video codec

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen

op donderdag 15 mei 2014 om 10:00 uur

door

Muhammad NADEEM

Master of Science in Electronics
Quaid-i-Azam University, Islamabad

geboren te Multan, Pakistan

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr. K.L.M. Bertels

Copromotor:
Dr.ir. J.S.S.M. Wong

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr. K.L.M. Bertels	Technische Universiteit Delft, promotor
Dr.ir. J.S.S.M. Wong	Technische Universiteit Delft, copromotor
Prof.dr. L. Carro	UFRGS Porto Alegre, Brasil
Prof.dr.ir. G.J.T. Leus	Technische Universiteit, Delft
Prof.dr.-Ing. M. Hübner	Ruhr-Universität Bochum, Germany
Prof.dr. Jose Pineda de Gyvez	Technische Universiteit Eindhoven
Dr. G.K. Kuzmanov	ARTEMIS-JU, Belgium
Prof.dr. P.M. Sarro	Technische Universiteit Delft, reservelid

This thesis has been completed in partial fulfillment of the requirements of the Delft University of Technology (Delft, The Netherlands) for the award of Ph.D. degree. The research described in this thesis was supported in parts by:
(1) CE Lab. Delft University of Technology,
(2) HEC Pakistan

Published by Muhammad Nadeem Email: muhammad.nadeem@hotmail.com

ISBN: 978-94-6186-301-0

Keywords: Low-power Architectures, VLIW processor, Softcore Processor, Application-specific Custom Instructions, Embedded Multimedia Systems, Reduced Complexity Video Compression Algorithms, Video Codec H.264/AVC, Run-time Adaptive Processor.

Printed in The Netherlands

Dedicated to my

Parents, Teachers, Wife and Kids

with gratitude and love

Adaptive, Low-power Architectures for Embedded Multimedia Systems

with focus on H.264/AVC video codec

Summary

RECENT advances in computing and communication technology have expanded the boundaries of communication systems to include a rich visual dimension. Standards-based video compression has played a significant role in the realization of these technologies by bridging the gap between demand for quality and performance, and limitations of current storage and transmission capabilities. Advanced video codecs, however, consume a significant amount of time and energy/power due to their adaptive processing nature in order to provide better compression. Moreover, the requirement for computing platforms to flexibly adapt to standard evolutions along with market and/or technology induced changes has added a new dimension to the processor architecture design. Consequently, power consumption and flexibility beside high-throughput have emerged as important design constraints in today's embedded multimedia systems.

Design methodologies have to take the power consumption into account at different levels of abstraction such as systems, architectures, algorithm/logic designs, basic cells as well as layouts. Efforts to reduce power consumption at higher abstraction levels are relatively more effective. However, these efforts may result in a system with less functionality or less programmability. Moreover, for embedded multimedia systems, the existing low-power design approaches generally tradeoff throughput and/or quality level.

In this dissertation, we target high-performance in terms of low-power, high-throughput, area-efficient, and flexible digital signal processing for battery-powered multimedia embedded systems, with a case study of H.264/AVC video codec. The goal of high-performance is achieved by exploiting the potential of area, power reduction and throughput enhancement at the algorithm design as well as at the processor architecture levels without compromising the video quality. From the algorithm design level perspective, we propose optimized video processing algorithms in H.264/AVC video codec for embedded

multimedia systems. The optimizations are essentially about algorithmic complexity reduction by removing redundancy in number of operations, extracting more parallelism, identifying mutually exclusive processing blocks within the algorithms, simplifying complex operations, re-using the intermediate results effectively, adapting for the statistical nature of video signal in algorithm design, etc. Efficient hardware designs for these optimized algorithms are then proposed by using standard low-power hardware design techniques such as clock-gating and/or data-enabling etc.

Traditional Von Neumann architecture based approach provides flexibility through software programming. This approach was based on the assumptions of hardware being expensive and the power consumption being non-critical. Therefore, time multiplexing approach was used to provide maximum sharing of the hardware resources. The situation, however, has fundamentally changed now. With potentially large number of cores possible on a chip, hardware has rather become a cheap commodity. Meanwhile, Reconfigurable Computing (RC) has emerged as a new paradigm for satisfying the simultaneous demand for application performance and flexibility. The dynamically reconfigurable processors have further boosted the dramatic nature of reconfigurable computing systems by combining programmability with adaptivity. This approach allows applications to be developed at high-level while at the same time, the processor organization can be adapted to application-specific requirements at design as well as at run-time. We believe that reconfigurable processors along with the application-specific data-paths can provide the needed flexibility and performance in energy-constrained embedded systems. In this dissertation, we identify various H.264/AVC video codec specific custom operations, propose their designs and implement them in the data-path of a dynamically reconfigurable, extensible, soft-core VLIW processor (ρ-VEX).

Adaptive, Low-power Architectures for Embedded Multimedia Systems

with focus on H.264/AVC video codec

Samenvatting

RECENTE ontwikkelingen in de informatica-en communicatietechnologie hebben de grenzen van communicatiesystemen verlegd om een rijke visuele dimensie toe te voegen. Standaard gebaseerde video compressie heeft een belangrijke rol gespeeld in de realisatie van deze technologieën door de kloof tussen de vraag naar kwaliteit en prestaties te verkleinen, en door beperkingen van de huidige opslag en het transport mogelijkheden boven te komen. Geavanceerde video codecs verbruiken echter een significante tijdshoeveelheid en energie / vermogen door hun adaptieve bewerkingsaard om betere compressie te leveren. Bovendien heeft de eis voor computerplatforms om flexibel te kunnen aanpassen aan standaard evoluties naast markt en / of technologie geïnduceerde veranderingen een nieuwe dimensie toegevoegd aan het processor architectuur ontwerp. Zodoende zijn energieverbruik en flexibiliteit naast high throughput naar voren gekomen als belangrijke ontwerp randvoorwaarden in huidige embedded multimediasystemen.

Ontwerpmethoden moeten rekening houden met het stroomverbruik op verschillende abstractieniveaus zoals systemen, architecturen, algoritme / logische ontwerpen, basiscellen en layouts. Inspanningen om het energieverbruik te verlagen op hogere abstractieniveaus zijn relatief effectiever. Daarentegen kunnen deze inspanningen resulteren in een systeem met minder functionaliteit of minder programmeerbaarheid. Bovendien maken de bestaande low-power ontwerpbenaderingen voor embedded multimedia-systemen in het algemeen een afweging voor throughput en / of kwaliteit.

In dit proefschrift richten we op high-performance in het gebied van low-power, high throughput, oppervlakte efficiënte en flexibele digitale signaalverwerking voor batterij gevoede multimedia embedded systemen, met een casestudy van H.264/AVC video codec. De high-performance target wordt bereikt door het exploiteren van het potentieel van de oppervlakte, vermogensvermindering en throughput verbetering van het algoritme ontwerp als op de

processorarchitectuur niveaus zonder afbreuk te doen aan de beeldkwaliteit. Vanuit het algoritme ontwerpniveau perspectief, stellen wij geoptimaliseerde video processing algoritmes voor in H.264/AVC video codec voor embedded multimedia systemen. De optimalisaties gaan in essentie over algoritmische complexiteit vermindering door het verwijderen van redundantie in vele operaties, het verkrijgen van meer parallellisme, het identificeren van elkaar uitsluitende verwerkingsblokken binnen de algoritmen, het vereenvoudigen van complexe operaties, tussentijdse resultaten effectief hergebruiken, aanpassing voor de statistische aard van videosignaal in algoritmeontwerp, enz. Efficiënte hardware ontwerpen voor deze geoptimaliseerde algoritmes worden vervolgens voorgesteld door gebruik te maken van standaard low-power hardware ontwerptechnieken zoals clock-gating en / of data-enabling enz. .

De traditionele Von Neumann architectuur gebaseerde methode biedt flexibiliteit aan door middel van het programmeren van software. Deze methode is gebaseerd op de aannames dat hardware duur is en dat het stroomverbruik niet kritisch is. Daarom werd tijd-multiplexing gebruikt om het maximaal delen van hardware resources mogelijk te maken. De situatie is nu echter fundamenteel veranderd. Met potentieel grote aantallen cores op een chip, is hardware uitgegroeid tot een goedkope grondstof. Ondertussen heeft Herconfigureerbaar Computing (RC) zich ontpopt als een nieuw paradigma voor het voldoen aan de gelijktijdige vraag naar applicatie performance en flexibiliteit. De dynamisch herconfigureerbare processoren hebben de dramatische aard van herconfigureerbare computersystemen verder versterkt door het combineren van programmeerbaarheid met adaptiviteit. Deze aanpak maakt het mogelijk om applicaties te ontwikkelen op hoog niveau, terwijl tegelijkertijd de processor organisatie kan worden aangepast aan de applicatie-specifieke eisen in de ontwerpfase en tijdens de uitvoering. Wij geloven dat herconfigureerbare processoren samen met applicatie-specifieke datapaden de benodigde flexibiliteit en prestaties in energie-gelimiteerde embedded systemen kunnen leveren. In dit proefschrift identificeren we verschillende H.264/AVC video codec specifieke aangepaste activiteiten, stellen we hun ontwerpen voor en implementeren hun in de datapad van een dynamisch herconfigureerbare, uitbreidbare, soft-core VLIW processor (ρ-VEX).

Acknowledgments

In the name of Allah the most gracious, the most merciful

First and foremost, I express my utmost gratitude to Almighty Allah SWT for endowing me with health, patience and knowledge to complete this dissertation.

This dissertation at hand, and the subsequent PhD has been a long journey full with moments of joy, sorrow, hard work, hopes, determination and conscientious efforts over many years. I am happy that at any rate, I am done. This dissertation would not have been possible and I could not have succeeded and gotten to where I am today, without the invaluable support of a several. I would like to take the opportunity to thank them all.

I owe my deepest gratitude to Prof.dr. Stamatis Vassiliadis for believing in me and accepting me in his group for doing my PhD. Though we spent very little time together and had only few meetings in Delft, but this is a fact that it was he who played a very important role in the process of making a decision to start this journey towards PhD. I am also deeply indebted to my supervisor Dr.ir. J.S.S.M. Wong. He always gave me the freedom to do whatever I wanted, at the same time he continued to contribute valuable feedback, advice, and encouragement. He never judged nor pushed when he knew I needed to juggle priorities. I also enjoyed working with Dr. G.K. Kuzmanov. The review comments on my draft papers, from Georgi, helped me a lot to improve my writing skills. My work would not have been the same without inputs from Stephan Wong and Gerogi Kuzmanov. I am also grateful to my promoter, Prof.dr. K.L.M. Bertels, especially for his help at the final stages of my PhD. I would like to extend my gratitude to the PhD examination committee for taking time out to read my dissertation and providing valuable comments. I am grateful to HEC and NUFFIC for theirfunding and support. I also express my sincere thanks to all the CE staff namely Lidwina for her administrative assistance, and to Bert, Erik and Eef for their technical support throughout these years.

During my stay in Netherlands and Belgium, I met many wonderful people. Most of them were very friendly, social and supportive, and some of them were a little reserve, conservative and shy too. Nevertheless, it was a great learning experience of my life. I would like to thank all my colleagues at Computer Engineering Laboratory. It was really an international environment with people from many different parts of the world with different background,

culture and faith. My appreciation also goes to my colleagues at NXP Software at Eindhoven, Mapper Lithography at Delft and Cochlear at Mechelen. They are so many that its simply not possible to mention them all here.

My appreciation goes to my Pakistani friends Hisham, Mehfooz, Laiq, Mazhar, Zubair, Husnul Amin, Tariq, Ahsan, Faisal, Fakhar, Seyab, Shah Muhammad, Aqeel, Imran, Umar, and Shahzad for their help to settle down here in Netherlands in my early days, for various family gathering and social events on the weekends to make me feel at home during my entire stay, and making my stay comfortable in my last days at Delft. Special thanks to Motta for translating my propositions and summary into Dutch. I am indebted to Faisal, for his valuable suggestions during proofreading of my dissertation. I would also like to take this opportunity to express my sincere appreciation to all my teachers who have taught me since I went to school.

Finally, There are no words that can express my gratitude for all what my parents have done for me. There is no doubt in my mind that I would not have been able to achieve this all, without their end less love, care, prayers, support, devotion and so many sacrifices throughout my life. I am also grateful to my brothers and sisters for all their kindness, support and encouragement.

Last but certainly not least, my tremendous, deep and especial thanks goes to my wife Amna and my kids Aisha, Abdullah and Khudaija. With my wife Amna, we have laughed and cried, traveled and played, built and settled, and planned and discussed our lives. I could not have completed this journey without her by my side. Aisha, Abdullah and Khudaija - happy, loving, and fun to be with kids deserve a special thanks for inspiring and amazing me every day. Although they often had to endure my absence but they seldom complained. I am sure they will enjoy the effort being completed.

Muhammad Nadeem
Delft, The Netherlands, 2014

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms and Symbols

ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-set Processor
BRAM	Block RAM
CABAC	Context Adaptive Binary Arithmetic Coding
CAVLC	Context Adaptive Variable Length Coding
CFU	Custom Functional Unit
CIF	Common Intermediate Format
DCT	Discrete Cosine Transform
DPR	Data Processing Rate
DSC	Digital Still Cammera
DSP	Digital Signal Processor
FIR	Finite Impulse Response filter
FPGA	Field Programmable Gate Arrays
GPP	General Purpose Processor
HDTV	High Definition TV
ILP	Instruction Level Parallelism
ISA	Instruction Set Architecture
JPEG	Joint Picture Expert Group
JVT	Joint Video Team
MPEG	Motion Picture Expert Group
RC	Reconfigurable Computing
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Logic
SAD	Sum of Absolute Difference
SATD	Sum of Absolute Transformed Difference
VCD	Value Change Dump
VCL	Video Coding Layer
VEX	VLIW Example
VHDL	Very High Scale Integrated Circuits Hardware Description Language
VLIW	Very Long Instruction Word

1

Introduction

DIGITAL video coding is one of the primitive applications of multimedia embedded systems. With the evolution of context-aware processing in advanced video coding standards, the exploitation of parallelism is becoming increasingly challenging [3] [4]. Advanced video codecs may consume a significant amount of processing time and energy/power, due to their adaptive nature of processing, to provide better compression. Beside performance and power consumption, other parameters like cost, short-time-to-market, mass volume production, flexibility and re-useability have created a multidimensional pressure on industry, as well as on research to come up with innovative architectures for embedded multimedia systems. The realization of advanced video coding with high-resolution videos on battery-powered mobile devices demands high complexity reduction in video coding algorithms for their real-time and low-power implementation. Consequently, the main objectives of this dissertation are to investigate how to achieve high-performance for real-time video processing applications in terms of throughput, while utilizing less on-chip resources. Similarly, this dissertation also proposes adaptive, low-power hardware design for multimedia video compression applications on battery-powered electronic devices without compromising the video quality.

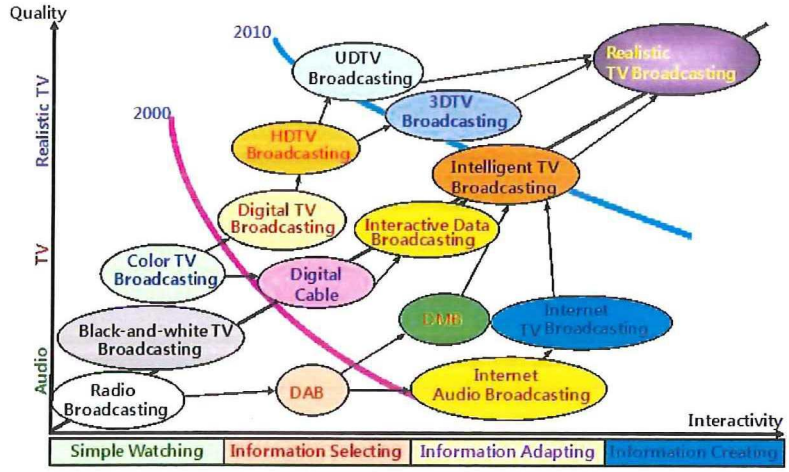
This introductory chapter starts with providing a brief overview on current trends in digital video processing applications/services and state-of-the-art in video compression technologies (Section 1.1). The hardware design options and trends in state-of-the-art for implementation of multimedia embedded systems along with the associated issues and challenges are summarized in Section 1.2. The summary of research challenges and goals is provided in Section 1.3. Similarly, Sections 1.4 and 1.5 describe the methodology and thesis contributions respectively. Finally, Section 1.6 presents organization of this dissertation.

1.1 Digital Video Coding Technology: Trends and Requirements

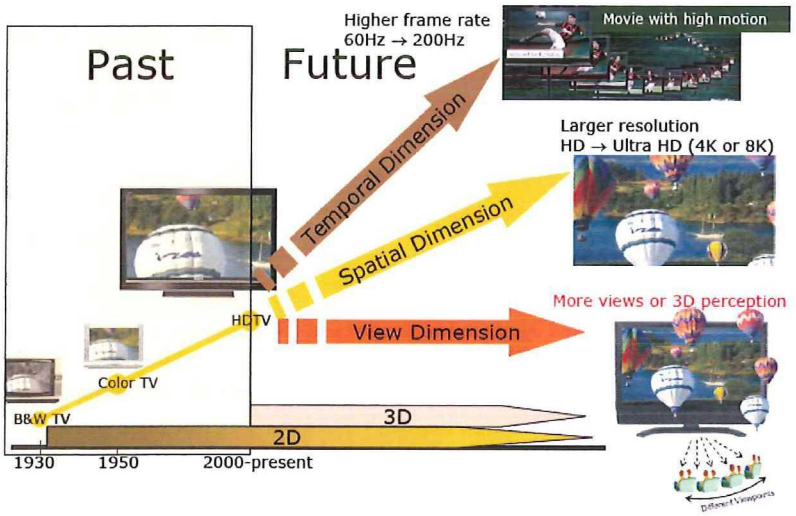
Why Video Compression? A video CODEC (enCOder / DECoder pair) is a software component or hardware device that enables compression and decompression of digital video signal. Typically, videos must be encoded before their storage or transmission, and decoded before their display. The amount of uncompressed video data is too large for limited transmission bandwidth or storage capacities. For example, to store a 2 - hours uncompressed video of full High Definition (HD) resolution frames (1920×1080 pixels), a capacity of 2 (hours) $\times 60$ (minutes per hour) $\times 60$ (seconds per minute) $\times 25$(frame rate, frames per second) $\times 1920 \times 1080$ (frame size in pixels) $\times 3/2$ (number of bytes per pixel) = $559,872$ GB is required, which far exceeds the capacity of current optical disc storage (50 GB for dual layer Blue-ray Disc). With efficient compression techniques, a significant reduction in amount of video data can be achieved with little or no adverse effect on the visual quality.

***Digital Video Services/Trends*:** Digital video coding is an enabling technology and generating ever new applications with a broadening range of requirements regarding basic video characteristics, such as spatio-temporal resolution, chroma format, and sample accuracy. Digital video-based services have become an integral part of wide range of industries–from telecommunications to broadcasting and entertainment to consumer electronics. Today, the application areas for digital video coding technology range from videoconferencing over mobile TV and broadcasting of standard-/ high-definition TV content, up to very high-quality applications such as professional digital video recording or digital cinema/large-screen digital imagery as illustrated in Figure 1.1a. Recently, the demands for more realistic multimedia contents have been increased dramatically. The ultimate objective of future video processing technology is to offer realistic 3D visual experiences. Full high-definition television technology has been adopted in consumer electronics, but it cannot offer sufficient realism yet. Therefore, attempts are being made to not only provide even higher-resolution images in spatial dimension (HD to Ultra HD), but also higher frame-rates up to 200 Hz (temporal dimension) and realistic 3D perception with more views (Figure 1.1b) [5].

Similarly, recent advances in computing and the communication technology have led to a significantly increased use of a large number of multimedia computing devices like mobile phones, personal navigation devices (PND), personal multimedia players (PMP), mobile internet devices (MID), tablets and



(a) Digital video services evolution over time [5]



(b) Digital video technology: past, present and future [5]

Figure 1.1: Digital video technology trends

notebooks. The average smart-phone usage grew 81% in 2012, whereas the mobile video traffic was more than 51% of the total global mobile data traffic by the end of same year [6]. The convergence of mobile phone, internet, mapping, gaming, and office automation tools with high-quality video and still imaging capture capability are becoming a strong market trend for portable devices. High-density video encode and decode, 3D graphics for gaming, increased application-software complexity, and ultra-high-bandwidth 4G modem technologies are driving the CPU performance and memory bandwidth requirements close to the PC segment. Consequently, on the one hand, the next generation multimedia applications being executed on these portable multimedia devices are becoming increasingly complex and consume more power to fulfill the end-user requirements. On the other hand, the growth in rechargeable battery-capacity has shown modest increases. Thus, the larger demand for increased functionality and speed has increased the severity of the power constraint in the world of hand-held and mobile multimedia systems. Even in the case of applications that have an unlimited energy source, we have moved into an era of power-constrained performance since heat removal requires the processor to operate at lower clock rates than dictated by the logic delays.

***Standard Based Video Coding*:** Different compression technologies, both proprietary and industry standards, are available. Video codec standards are important in ensuring compatibility and interoperability. Over the last two decades, standard based digital video coding/compression technologies have evolved from MPEG-1 to MPEG-2/H.262 to H.264/AVC. The H.264/AVC [7] is the latest video coding standard jointly developed by ISO-IEC and ITU-T. It provides $2\times$ compression compared to previous coding standards (such as H.262/MPEG-2, H.263) for the same subjective video quality at the cost of additional computational complexity and energy consumption (approx. $10\times$ relative to MPEG-4 advance simple profile [8]). Beside higher resolutions, the key reason of increasing video coding complexity is the complex tool-set for advance video encoders. The authors in [1] state an expected increase in the video complexity by $2\times$ every two years (Figure1.2). Although, high resolutions are mainly targeted for high-end multimedia devices, multiview video conferencing or personal recording at much higher resolutions Quad-HD, 3840×2160 or 4096×2304) is foreseen within next few years on mobile devices. Industrial prototypes like [9] have already demonstrated the feasibility of 3D-videos and multiview video coding on mobile devices using two views. In short, as mentioned before, with the evolution of context-aware processing in advanced video coding standards, exploitation of parallelism is becoming extensively challenging [3], [4]. Moreover, besides high performance in terms

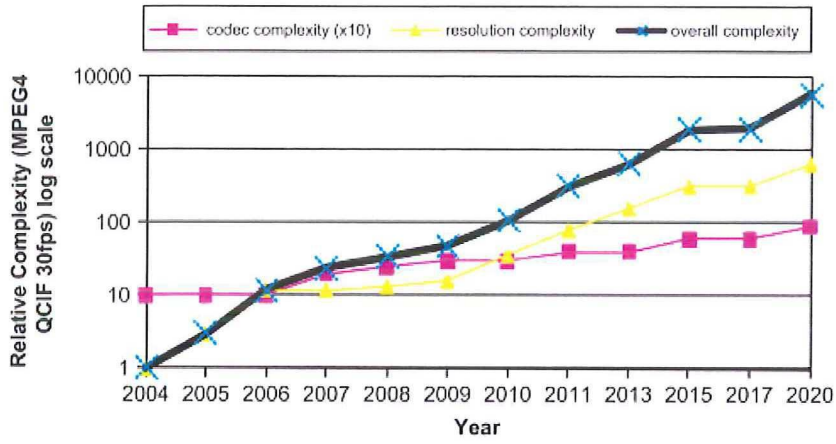


Figure 1.2: Codec, Resolution complexity relative to QCIF 30fps [1]

of throughput, the realization of advanced video coding with high resolution videos on battery-powered mobile devices demands high complexity reduction in video coding algorithms for their low-power implementation.

1.2 Multimedia Embedded Systems: Options and Challenges

Traditionally, **standard-cell ASICs** are considered as the best implementation choice for high-volume, price-sensitive applications–especially in the consumer electronics market. ASICs offer the highest performance possible for most of the applications. Since the ASICs target a specific application, therefore, they provide smaller form-factor and can be specifically optimized for *performance per unit area* and *performance per unit power consumption* for the target application. Although, structured ASICs have some limited design flexibility, however, ASICs lack flexibility and adaptability in general and therefore, hard to adapt to standard evolutions and market/technology induced changes. Video codec H.264/AVC, for instance, provide a large set of tools to support a range of applications (e.g., low bit-rate video conferencing, high-quality personal video recording, HDTV, etc.). A generic ASIC for all tools is impractical and will be huge in size. In contrast, multiple ASICs for different applications have a longer design time and thus an increased Non-Recurring Engineering (NRE) cost.

Digital Signal Processors (DSPs), on the other hand, are programmable and

offer high flexibility over ASICs. DSPs include support for frequently used signal processing operations and addressing modes. These operations include multiple parallel multiply-accumulate operations used to implement efficiently computation kernels, such as FIR/IIR filters and linear transformation operations. The specialized ISA and specialized functional units, therefore, enable DSPs to provide better *performance per unit area* and *performance per unit power consumption* over General Purpose Processor (GPPs) for a software-based multimedia system. However, DSPs alone may not satisfy the power and/or performance challenges when considering the combination of tight power budgets on battery-powered mobile devices and intricate processing nature of next-generation multimedia algorithms. Moreover, DSP performance is limited by the available data bandwidth from the external memory [10], [11]. Although stream architecture [10] provides an efficient memory hierarchy to exploit the concurrency and data locality, it exploits a limited amount of parallelism (e.g., only data level parallelism) [12]. DSP based software solutions alone do offer flexibility but can not provide the required performance and, therefore, dedicated hardware accelerators are often inevitable.

ASIPs can be tailored for a specific application and, therefore, provides a tradeoff between flexibility and performance of some degree. They overcome the shortcomings of DSPs and ASICs and provide better *performance per unit area* and *performance per unit power,* when compared with GPP and DSPs. A number of IPs for embedded customizable processor along with tool suites are available from vendors like Tensilica and ARC. Application specific instruction set processors (ASIPs) offers a compromise between flexibility and performance, however, the customization of ASIP for larger applications with many kernels may end up with considerably larger core size and, therefore, requires larger silicon footprint.

MPSoCs, on the other hand, deliver high performance and programmability by integrating application-specific hardware accelerators with programmable processors like GPPs and DSPs. Commercially available OMAP from Texas Instruments, Trimedia/Nexperia from Philips/NXP and Nomadik by STMicroelectronics are some prominent examples of MPSoCs. Since the programmable cores and application-specific hardware accelerators are selected at design time targeting certain type of applications. Therefore, MPSoC may not provide sufficient performance for applications from different domains. Moreover, the currently-used MPSoC may becomes obsolete because of need to support a new standard or evolutions in the current standard..

Reconfigurable Computing (RC) is emerging as the new paradigm for sat-

isfying the simultaneous demand for application performance and flexibility. Reconfigurable computing utilizes hardware that can be adapted at run-time to facilitate greater flexibility without compromising performance. It combines the flexibility of software with the high performance of hardware by processing with very flexible high speed computing fabrics like FPGAs. The principal difference when compared to using ordinary microprocessors is the ability to make substantial changes to the data-path itself in addition to the control flow. On the other hand, the main difference with custom hardware (ASICs) is the possibility to adapt the hardware during run-time by *loading* a new circuit on the reconfigurable fabric. Reconfigurable architectures can exploit fine-grain and coarse-grain parallelism available in the application because of the adaptability. Exploiting this parallelism provides significant performance advantages compared to conventional microprocessors. The re-configurability permits adaptation of the hardware for specific computations in each application to achieve higher performance compared to software. Complex functions can be mapped onto the architecture achieving higher silicon utilization and reducing the instruction fetch and execute bottleneck.

1.3 Research Challenges

In previous sections, we argued that even though processor speeds and network bandwidth continue to increase, efficient video processing for multimedia consumer electronics devices is still a very important requirement for a successful/competitive product. Consequently, in this dissertation we shall investigate:

- *How to achieve high-performance and flexible processing in video processing/compression applications?*

Since H.264/AVC is state-of-the-art in video compression standards and has already been widely accepted by the industry. Consequently, we shall focus on how to improve the performance of compute-intensive modules in H.264/AVC video codec in terms of throughput, resources/area-cost and power-consumption. Our first problem statement is:

- *How to reduce the coding complexity and improve the real-time performance without compromising the video quality?*

Like previous video coding standards (H.261, H.262/MPEG-2, H.263 etc.), H.264/AVC also does not explicitly define a CODEC (enCOder / DECoder

pair). Rather, it defines the syntax of an encoded video bitstream together with methods for the decoding process of these syntax elements. There are a couple of syntax elements, where a set of constraints is imposed by the video coding standard and the actual method/algorithm to compute these syntax elements is out of scope of the standard. Whereas, for couple of other compute-intensive syntax elements, an algorithm/method along with constraints is specified in the H.264/AVC video coding standard. This dissertation shall focus on the complexity reduction of standardized video coding algorithms/methods.

The tradeoff of various types of architectures to implement DSP algorithms has been a topic of research since the initial development of the theory. Recently, the application of these DSP algorithms to systems those require low cost and the lowest possible energy consumption has placed a new emphasis on defining the most appropriate solutions. Moreover, the flexibility requirement to adapt to standard evolutions and market/technology induced changes has become a new dimension in the algorithm/architecture design. Traditional Von Neumann architecture-based approach, provides flexibility through software programming. This approach was based on technology assumptions that hardware was expensive and the power consumption was not critical. Therefore, time multiplexing approach was used to provide maximum sharing of the hardware resources. The situation, however, is fundamentally different now. The energy/power consumption is a critical design constraint beside throughput in battery-powered portable devices. Whereas, with potentially thousands of multipliers and adders available on a chip, hardware is rather a cheap commodity. Similarly, even in the case of applications with an unlimited energy source, we have moved into an era of power-constrained performance since heat removal requires the processor to operate at lower clock rates than dictated by the logic delays. Consequently, in relation with above mentioned first problem statement, this dissertation shall:

- *Propose new algorithms or optimizations for existing algorithms within the scope of standardized video coding blocks/components (H.264/AVC) to achieve complexity reduction without compromising the video quality.*
- *Design, implementation, and evaluation of high-throughput, area-efficient and low-power video signal processing architecture for various video processing components/tools in H.264/AVC.*

As mentioned before, the flexibility requirement to adapt to video coding standard evolutions and market/technology induced changes has become a new dimension in the algorithm/architecture design. We believe that reconfigurable

computing could possibly be the solution to provide the needed flexibility beside performance. Our second problem statement is:

- *How can reconfigurable computing be utilized to improve the flexibility and performance for video coding signal processing applications?*

Reconfigurable computing has proven itself to be able to speed-up many applications despite its lack in achieving high frequencies. However, frequency is not the sole factor that determines performance. Field-programmable Gate Arrays (FPGAs) - as the most utilized reconfigurable fabric nowadays - provide a large amount of parallel structures that when exploited efficiently can greatly contribute to the speedup of applications.

Digital video coding/signal processing applications have been identified to have significant fine- and coarse-grained parallelism [13] Therefore, Very Long Instruction Word (VLIW) processors can be utilized to increase the performance beyond normal Reduced Instruction Set Computer (RISC) architectures [14]. While RISC architectures only take advantage of temporal parallelism (by utilizing pipelining), VLIW architectures can additionally take advantage of the spatial parallelism by utilizing multiple functional units (FUs) to execute several operations simultaneously. VLIW processor improves the performance by exploiting Instruction Level Parallelism (ILP) in a program. Similarly, the customization of processors for an application is another way of improving the performance for a moderate cost. Consequently, in relation with our second problem statement, this dissertation shall:

- *Propose customization of reconfigurable VLIW processor for compute-intensive H.264/AVC video codec tools to improve the performance.*

1.4 Methodology

From digital video coding perspective, with a wide range of target applications from low-end to high-end under various constraints, such as throughput, power consumption and resources/area cost, an application-specific implementation for digital video coding algorithms, may be pure software, pure hardwired, or something in between. The video coding algorithm defines a detailed implementation outline of a required original function and, therefore, determines how to solve the problem and how to reduce the original complexity. In order to do an optimal implementation, it is essential to fully understand the principles behind and algorithms employed in video coding as it is a key to reduce

power consumption and improve efficiency. In this section, we propose different steps to achieve high-performance video processing systems as proposed in the previous section. These steps are listed in the following:

- *Identify the compute-intensive video compression tools within the scope of video coding standard H.264/AVC.*
- *Propose algorithmic optimization to reduce complexity of the video compression algorithms.*
- *Exploit video signal statistics and data correlations for further complexity reduction.*
- *Propose efficient design in terms of high-throughput, low-area and low-power consumption, for compute-intensive processing units in H.264/AVC.*
- *Investigate the processing chain for video compression engine in H.264/AVC for possible complexity reduction by reusing the intermediate results and propose an efficient algorithm.*
- *While utilizing the reduced-complexity optimized algorithms, design, implement and evaluate the high-throughput, area-efficient and low-power solution for compute-intensive processing units in H.264/AVC.*
- *Validate the proposed design using variety of video test sequences from Joint Video Team (JVT), responsible for development of video coding standard H.264/AVC, and further comparing the results against that of reference video coding standard implementation provided by JVT.*

In order to support processor level adaptivity and customization, dynamically reconfigurable processor (ρ-VEX) shall be used as a target computing platform. ρ-VEX is an open source, extensible and reconfigurable soft-core VLIW processor. The processor architecture is based on the VEX (VLIW Example) Instruction Set Architecture (ISA), as introduced in [15], and is implemented on an FPGA. The ISA of ρ-VEX shall be extended with custom operations. From the prospective of our second problem statement, we propose to combine multiple-issue architectures (ρ-VEX) and customization techniques to further improve the performance of the processor.

- *Design, implement, and evaluate custom operations for compute-intensive processing tools/units within scope of video coding standard H.264/AVC.*

1.5 Research Contributions

The research carried out in the course of this PhD project is published in several scientific publications. This section highlights the main contributions of the research work described in this dissertation, as follows:

1. In-loop deblocking filter.

    - The complexity of in-loop deblocking filter algorithm is reduced by novel decomposition of filter kernels and intra-module optimizations. The complexity of the deblocking filter algorithm is reduced by more than 51% when compared with that of algorithm described in the video coding standard H.264/AVC [16].
    - A low-power hardware design for deblocking filter unit is proposed. Experimental results suggest that the dynamic power consumption is reduced up to 50%, when compared with state-of-the-art designs in literature. [17].
    - For real-time video processing applications, a high-throughput, area-efficient hardware design, based on the low-power filter unit, is proposed. . This design utilizes 2 filter units and, therefore, can process input pixels on-the-fly in both horizontal and vertical directions. The proposed design provides significantly higher throughput (more than 63% when compared with state-of-the-art in literature having similar area-cost) and require less on-chip area (around 35% when compared with state-of-the-art in literature having similar throughput) [16].

2. Intra-prediction

    - The complexity of intra-prediction algorithm for various intra-prediction modes is reduced by 27% - 60% in comparison with intra-prediction algorithm proposed in H.264/AVC video coding standard. A configurable, high-throughput, and area-efficient hardware design for intra-prediction unit, based on the reduced complexity intra-prediction algorithm, is proposed in this dissertation. The comparison with other state-of-the-art suggests that our proposed hardware design provides 50%-75% performance improvement and requires only 21K gates for its implementation, when synthesized under 0.18 μm CMOS standard cell technology [18].

3. Forward integer transform

   - A transformation, to compute integer-transformed residual block from Hadamard-transformed residual block in the processing chain of H.264/AVC video codec, is proposed in this dissertation. This approach reduces the number of addition operations by more than 50% for realization of integer transform in the video codec. The comparison with the existing single 4×4 forward integer transform solutions from the literature suggest that the proposed solution provides the minimum latency penalty (4.82ns) among all solutions and also requires significantly less area (2.6K gates) in terms of equivalent gate count for its hardware implementation. Therefore, it provides up to 5 times better performance in terms of throughput/area ratio for the same process technology. [19].
   - Similarly, a low-latency and area-efficient solution for realization of forward integer transform unit in the intra-frame processing chain is proposed. With this proposed solution, the effective latency penalty for the forward integer transform unit is reduced to zero. In additions to zero latency penalty in the intra-frame processing chain, the proposed solution provides up to 30 times better performance in terms of throughput/area ratio. [19].

4. Inverse integer transform

   - For inverse integer transform unit, a configurable, low-power hardware design is presented in this dissertation. The proposed design is based on a data-driven computation algorithm for the inverse integer transform. It efficiently exploits the zero-valued coefficients in the input blocks to reduce dynamic power consumption. The experimental results show that the proposed design consumes significantly less dynamic power (up to 80% reduction), when compared with existing conventional designs for the inverse integer transform, with a small area-overhead (approximately 2 K gates) [20] [21].

5. Custom instructions/processing units for extensible, reconfigurable, soft-core VLIW processor (ρ-VEX).

   - In this dissertation, we also proposed custom operations for deblocking filter, intra-prediction and forward/inverse integer transform units in H.264/AVC foe a reconfigurable VLIW processor.

The datapath is based on the optimized algorithms presented in this dissertation. The proposed custom operations significantly reduce the compute time (approximately 40% - 59%) for the corresponding compute-intensive functions in H.264/AVC on a reconfigurable, soft-core VLIW processor (ρ-VEX). [8.9, 8.11, 8.13].

1.6 Thesis Organization

The remainder of the thesis is organized as follows.

Chapter 2 provides the background for digital video coding, with a focus on advanced video codec H.264/AVC. Next, the prominent related work on high-throughput, area-efficient and low-power codec design and implementations is presented. The background for dynamically reconfigurable VLIW processor (ρ-VEX) is also is presented towards end of the chapter.

Chapter 3 covers the deblocking filter in H.264/AVC. Low-power and high-throughput hardware designs based on single and double filter units are presented for image and real-time video processing applications, respectively.

Chapter 4 deals with the intra-prediction module in H.264/AVC. This chapter introduces an optimized algorithm to compute all intra-prediction mode for 4x4 pixel block along with the hardware design.

Chapter 5 introduces two designs for the realization of forward integer transform. A novel transform is also presented in this chapter to derive the forward integer transform coefficients directly from that of Hadamard transform coefficients.

Chapter 6 deals with inverse integer transform module in H.264/AVC. Two hardware designs for the inverse integer transform are presented in this chapter. The first design is intended for intra-frame encoder with reduced latency. The proposed design process the input data on-the-fly to produce the inverse transformed data block. A data-driven algorithm with variable number of operations is also introduced in this chapter. The second hardware design, based on the data-driven algorithm, provides high-throughput and consume significantly less dynamic power for its implementation.

Chapter 7, proposes to incorporate a customized functional unit in the data path of reconfigurable, soft-core, VLIW processor (ρ-VEX). This customized functional unit implements several application specific custom instructions for the compute intensive processing blocks in video codec H.264/AVC.

Chapter 8 provides experimental results for the proposed designs, in terms of throughput, area and dynamic power consumption. A comparison with the designs already presented in the literature is also done in the same chapter.

Finally, Chapter 9 concludes our work and also provides outlook of the potential future works.

2

Background and Related Work

VIDEO compression or video encoding is a process of reducing the amount of data required to represent a digital video signal, prior to transmission or storage. Similarly, video decoding recovers the digital video signal from the compressed representation, prior to display. In order to provide solutions of high quality (high frame resolution, high frame rate, and low distortion) or low cost (low bit rate for storage or transmission) or both, video compression is indispensable when storage capacity or transmission bandwidth is constrained. Advancement in semiconductor technology makes possible the efficient implementation of effective but computationally complicated compression methods.

This chapter provides the background information for the H.264/AVC video coding standard and introduces the functional blocks of video codec in Section 2.1. This section also identifies the compute-intensive functional units in H.264/AVC video codec. Section 2.2 introduces the ρ-VEX Reconfigurable VLIW processor. Finally, Section 2.3 presents the related work for the identified compute-intensive video coding functional units in H.264/AVC and Section 2.4 summarizes this chapter.

2.1 H.264/AVC Video Coding Standard Overview[1]

The H.264/AVC (also known as MPEG4 Part 10) is state-of-the-art video coding standard jointly developed by ITU-T and ISO/IEC. In common with earlier video coding standards from ITU-T (such as H.261 and H.263) and ISO/IEC (such as MPEG-1, MPEG-2, and MPEG-4), the H.264/AVC standard does not explicitly define a CODEC (enCOder / DECoder pair). Rather, the standard defines the syntax of an encoded video bit-stream together with the method of

[1]The overview of the H.264/AVC video coding standard is extracted from [2] and [7].

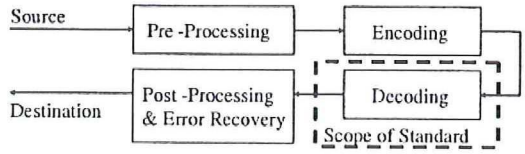


Figure 2.1: Scope of video coding standardization [2]

decoding process of the syntax elements. This is how the standard ensures that every decoder conforming to the standard will produce similar output when given an encoded bit stream that conforms to the constraints of the standard. The scope of the standardization is illustrated in Figure 2.1, which shows the typical video coding/decoding chain (excluding the transport or storage of the video signal). This limitation of the scope of the standard permits maximum freedom to optimize implementations in a manner appropriate to specific applications (balancing compression quality, implementation cost, time to market, etc.). However, it provides no guarantees of end-to-end reproduction quality, as it allows even crude encoding techniques to be considered conforming.

The H.264/AVC is a block-based hybrid video coding standard. The functional units in H.264/AVC codec are depicted in Figure 2.2. The main processing units in a video codec are, Prediction (intra-frame/inter-frame prediction), special Integer Transform, Quantization, Deblocking filter and Entropy Encoding. There is no single coding unit in the this video coding layer (VCL) that provides the majority of the significant improvement in compression efficiency in relation to prior video coding standards. It is rather a plurality of smaller improvements that add up to the significant gain.

Pictures, Frames, and Fields. A coded video sequence in H.264/AVC consists of a sequence of coded pictures. A coded picture in [7] can represent either an entire frame or a single field. Generally, a frame of video can be considered to contain two interleaved fields, a top and a bottom field. The top field contains even-numbered rows, whereas the bottom field contains the odd-numbered rows. If the two fields of a frame were captured at different time instants, the frame is referred to as an interlaced frame, and otherwise it is referred to as a progressive frame as depicted in Figure 2.3. The coding representation in H.264/AVC is primarily agnostic with respect to this video characteristic, i.e., the underlying interlaced or progressive timing of the original captured pictures. Instead, its coding specifies a representation based primarily on geometric concepts rather than being based on timing.

YCbCr Color Space and 4:2:0 Sampling. The human visual system seems to

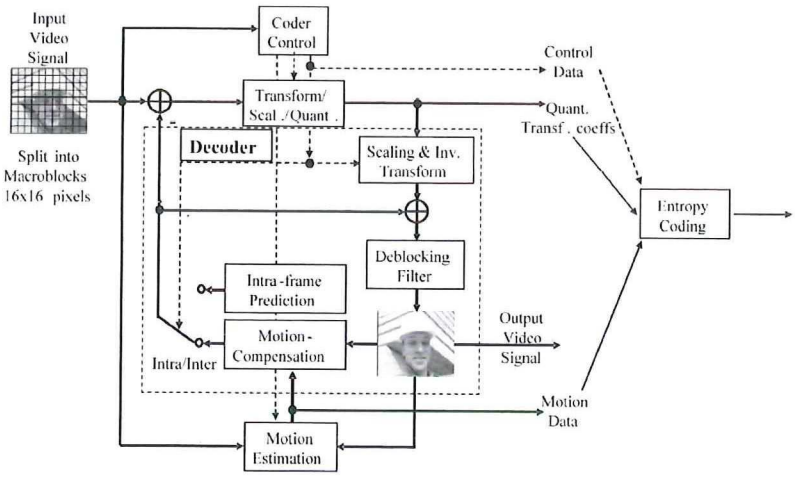


Figure 2.2: Basic coding structure for for a macroblock in H.264/AVC

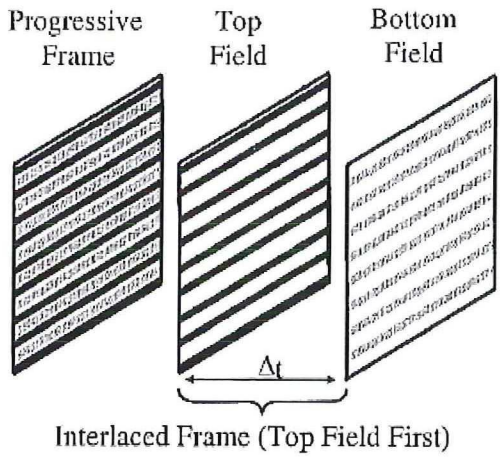


Figure 2.3: Progressive and Interlaced Frames and Fields

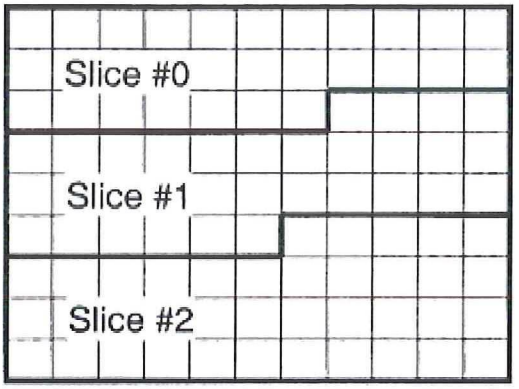


Figure 2.4: Subdivision of a picture into slices

perceive scene content in terms of brightness and color information separately, and with greater sensitivity to the details of brightness than color. The video color space used by H.264/AVC separates a color representation into three components called Y, Cb, and Cr. Component Y is called luma, and represents brightness. The two chroma components Cb and Cr represent the extent to which the color deviates from gray toward blue and red, respectively. Because the human visual system is more sensitive to luma than chroma, H.264/AVC uses a sampling structure in which the chroma component has one fourth of the number of samples than the luma component. This is called 4:2:0 sampling with 8 bits of precision per sample.

Slices and Macroblocks. A picture is partitioned into fixed-size macroblocks that each cover a rectangular picture area of 16×16 samples of the luma component and 8×8 samples of each of the two chroma components. Macroblocks are the basic building blocks of the standard for which the decoding process is specified. Slices are a sequence of macroblocks which are processed in the order of a raster scan. A picture maybe split into one or several slices as depicted in Figure 2.4. A picture is, therefore, a collection of one or more slices in H.264/AVC. Slices are self-contained in the sense that given the active sequence and picture parameter sets, their syntax elements can be parsed from the bit-stream and the values of the samples in the area of the picture that the slice represents can be correctly decoded without use of data from other slices. Some information from other slices maybe needed to apply the deblocking filter across slice boundaries. Each slice can be coded using different coding types as follows:

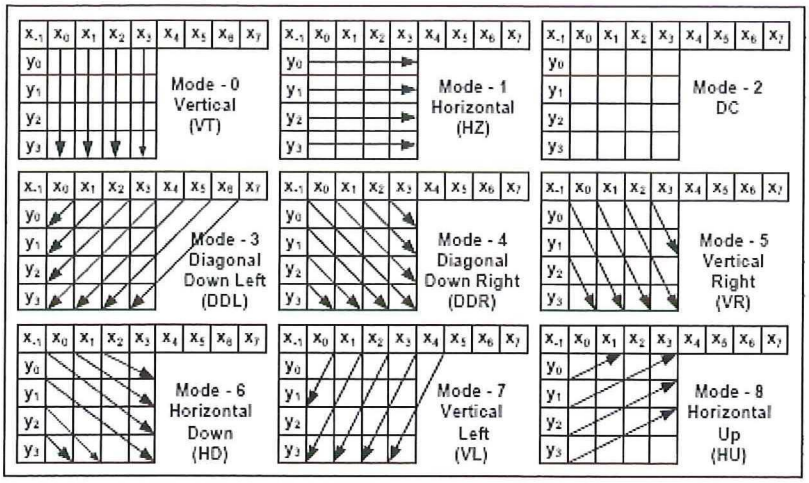


Figure 2.5: Illustration of nine 4×4 luminance prediction modes

- I-Slice: A slice where all macroblocks are encoded using predictions from neighboring macroblock within the same slice. (Intra-prediction).
- P-Slice: In addition to the coding types of the I-slice, some macroblocks of the P-slice can also be coded with at most one motion-compensated prediction signal per prediction block from external to the current slice in past video frame. (Inter-prediction).
- B-Slice: In addition to the coding types available in a P-slice, some macroblocks of the B-slice can also be coded with two motion-compensated prediction signals per prediction block from external to the current slice in past as well as future video frame. (Bi-directional inter-prediction).

Encoding and Decoding Process for Macroblocks. All luma and chroma samples of a macroblock are either spatially or temporally predicted, and the resulting prediction residual is encoded using transform coding (Integer transform + Hadamard transform). For transform coding purposes, each color component of the prediction residual signal is subdivided into smaller 4 × 4 blocks. Each block is transformed using an integer transform, and the transform coefficients are quantized and encoded using entropy coding methods.

Intra-Frame Prediction. The H.264/AVC supports multiple directional intra-prediction modes to reduce the spatial redundancy in the video signal. These multiple intra-prediction modes help to significantly improve the encoding performance of an H.264 intra-frame encoder. The Intra 4 × 4 mode is based on

predicting each 4 × 4 luma block separately and is well suited for coding of parts of a picture with significant detail. The Intra 16 × 16 mode, on the other hand, performs prediction of the whole 16 × 16 luma block and is more suited for coding very smooth areas of a picture. In addition to these two types of luma prediction, a separate chroma prediction is conducted. As an alternative to Intra 4 × 4 and Intra 16 × 16, the standard also provides an option to bypass the prediction and transform coding process and directly send the values of the encoded samples using Intra PCM coding type.

When using the Intra 4 × 4 mode, each 4 × 4 block is predicted from spatially neighboring samples as illustrated in Figure 2.5. The 16 samples of the 4 × 4 block are predicted using prior decoded samples in adjacent blocks labeled as x0 - x7 and y0 - y7. For each 4 × 4 block, one of nine prediction modes can be utilized. In addition to "DC" prediction (where one value is used to predict the entire 4 × 4 block), eight directional prediction modes are specified as illustrated in Figure 2.5. Those modes are suitable to predict directional structures in a picture such as edges at various angles.

When utilizing the Intra 16 × 16 mode, the whole luma component of a macroblock is predicted. Four prediction modes are supported. Prediction mode 0 (vertical prediction), mode 1 (horizontal prediction), and mode 2 (DC prediction) are specified similar to the modes in Intra 4 × 4 prediction except that instead of 4 neighbors on each side to predict a 4 × 4 block, 16 neighbors on each side to predict a 16 × 16 block are used. For the specification of prediction mode 4 (plane prediction), please refer to [7]. The chroma samples of a macroblock are predicted using a similar prediction technique as for the luma component in Intra 16 × 16 macroblocks, since chroma is usually smooth over large areas. Intra prediction across slice boundaries is not used, in order to keep all slices independent of each other.

Inter-Frame Prediction. In addition to the intra macroblock coding types, various predictive or motion-compensated coding types are specified as P macroblock types. Each P macroblock type corresponds to a specific partition of the macroblock into the block shapes used for motion-compensated prediction. Partitions with luma block sizes of 16 × 16, 16 × 8, 8 × 16, and 8 × 8 samples are supported by the syntax. In case partitions with 8 × 8 samples are chosen, one additional syntax element for each 8 × 8 partition is transmitted. This syntax element specifies whether the corresponding 8 × 8 partition is further partitioned into partitions of 8 × 4, 4 × 8, or 4 × 4 luma samples and corresponding chroma samples. Figure 2.6 illustrates the partitioning.

Transform, Scaling, and Quantization. Similar to previous video coding

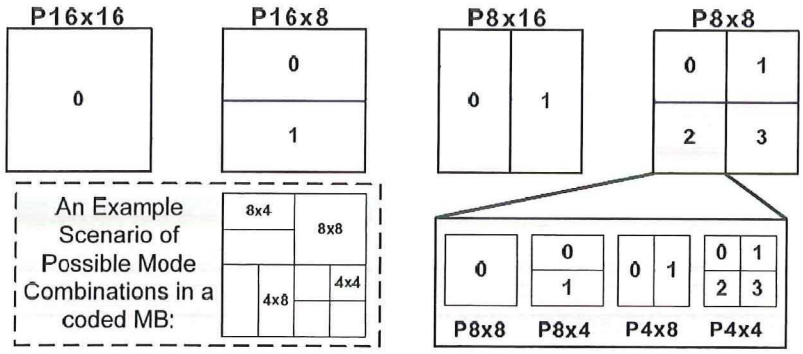


Figure 2.6: Variable block sizes inter-predicted MBs

standards, H.264/AVC utilizes transform coding of the prediction residual. However, in H.264/AVC, the transformation is applied to 4 × 4 blocks, and instead of a 4 × 4 discrete cosine transform (DCT), a separable integer transform with similar properties as a 4 × 4 DCT is used. The transform matrix is given as:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}$$

Since the inverse transform is defined by exact integer operations, inverse-transform mismatches are avoided. The basic transform coding process includes a forward transform, zig-zag scanning, scaling, and rounding as the quantization process followed by entropy coding. At the decoder, the inverse of the encoding process is performed except for the rounding. More details on the specific aspects of the transform in H.264/AVC can be found in [22].

A quantization parameter is used for determining the quantization of transform coefficients in H.264/AVC. The parameter can take 52 values. These values are arranged so that an increase of 1 in quantization parameter means an increase of quantization step size by approximately 12% (an increase of 6 means an increase of quantization step size by exactly a factor of 2). The quantized transform coefficients of a block generally are scanned in a zig-zag fashion and transmitted using entropy coding methods.

Entropy Coding. In H.264/AVC, two methods of entropy coding are supported. The simpler entropy coding method uses a single infinite-extent codeword table for all syntax elements except the quantized transform coefficients.

Thus, instead of designing a different VLC table for each syntax element, only the mapping to the single codeword table is customized according to the data statistics. The single codeword table chosen is an exp-Golomb code with very simple and regular decoding properties.

For transmitting the quantized transform coefficients, a more efficient method called Context-Adaptive Variable Length Coding (CAVLC) is employed. In this scheme, VLC tables for various syntax elements are switched depending on already transmitted syntax elements. Since the VLC tables are designed to match the corresponding conditioned statistics, the entropy coding performance is improved in comparison to schemes using a single VLC table. The efficiency of entropy coding can be improved further if the Context-Adaptive Binary Arithmetic Coding (CABAC) is used. Compared to CAVLC, CABAC typically provides a reduction in bit rate between 5%–15% [23].

In-Loop Deblocking Filter. One particular characteristic of block-based coding is the accidental production of visible block structures. Block edges are typically reconstructed with less accuracy than interior pixels and "blocking" is generally considered to be one of the most visible artifacts with the present compression methods. For this reason, H.264/AVC defines an adaptive in-loop deblocking filter, where the strength of filtering is controlled by the values of several syntax elements.

The basic idea of deblocking filter is that if a relatively large absolute difference between samples near a block edge is measured, it is quite likely a blocking artifact and should therefore be reduced. However, if the magnitude of that difference is so large that it cannot be explained by the coarseness of the quantization used in the encoding, the edge is more likely to reflect the actual behavior of the source picture and should not be smoothed over. The deblocking filter in H.264/AVC reduces the blockiness while keeping the sharpness of the content unchanged. Consequently, the subjective quality is significantly improved. The filter reduces the bit rate typically by 5%–10% while producing the same objective quality as the non-filtered video. A detailed description of the adaptive deblocking filter can be found in [24].

2.2 Overview of the ρ-VEX VLIW Processor [2]

ρ-VEX is an open source, extensible, and reconfigurable, soft-core VLIW processor. The processor architecture is based on the VEX (VLIW Example)

[2]The overview of the r-VEX VLIW is extracted from [25]

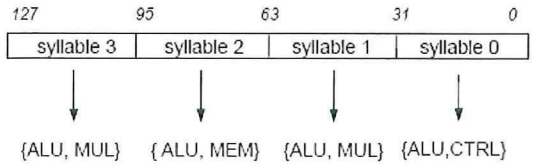


Figure 2.7: ρ-VEX instruction layout

Instruction Set Architecture (ISA), as introduced in [4], and is implemented on an FPGA. Parameters of the VLIW processor such as the number and type of functional units (FUs), supported instructions, memory bandwidth, and register file size can be chosen based on the application requirements and the available resources on the FPGA. A software development tool-chain including a highly optimizing C compiler and a simulator for VEX is made freely available by Hewlett-Packard (HP) [5]. Any application written in C can be executed on the processor implemented on the FPGA. The ISA can be extended with custom operations and the compiler is able to generate code for the custom hardware units, further enhancing the performance. The following parameters of our current design can be changed:

1. Issue-width of the processors pipeline
2. Type and the location of functional units
3. Size of the GP and BR register files
4. Size of the data and instruction memory
5. Presence of forwarding logic

The architecture and organization of the processor is provided in Sections 2.2.1 and 2.2.2 respectively.

2.2.1 ρ-VEX : Processor Architecture

The VEX ISA calls a encoded operation a syllable, and it defines an instruction as a set of syllables [3]. In VEX operations are equivalent to 32 bit RISC instructions. The 4-issue processor instance utilizes the instruction layout depicted in Figure 2.7. This figure illustrates how syllables are mapped to sets of FUs.

The ρ-VEX processor has a Load/Store Harvard architecture with multiple issue-slots. Its micro-architecture is pipelined with 5 stages. The processor has 4 types of FUs, namely: ALU, Multiplier, Load/Store, and Branch.

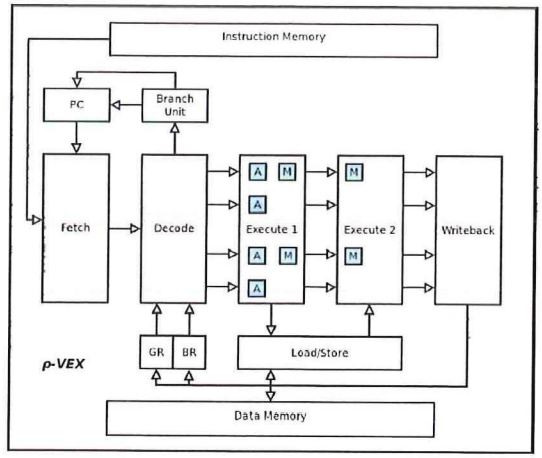


Figure 2.8: ρ-VEX pipeline organization

More specifically, there are four 32-bit ALUs, two 16 × 32 bit Multipliers, one Load/Store Unit, and one Branch Unit (BU). In addition, the architecture defines a 64 × 32- bit general-purpose register file, and a branch register file (BR) with 8 × 1-bit branch registers that store branch conditions, predicate values and the carries from arithmetic operations. The number of each type of FU, except for the BU's issue slot, is adjustable. This allows to customize the processor based on the the target application(s) requirements.

There are 70 native VEX operations divided over 5 classes: 39 ALU operations, 11 multiplication operations, 9 memory operations, 9 control operations, and 2 inter-cluster operations. ALU operations are partitioned into arithmetic and logical operations.

2.2.2 ρ-VEX : Processor Organization

The pipeline of all ρ-VEX processor instances consists of five stages: Fetch, Decode, Execute 1, Execute 2, and *WriteBack*. The organization of a 32-bit, four-issue ρ-VEX VLIW processor is depicted in Figure 2.8. The Fetch stage reads VLIW instructions from instruction memory, splits them into syllables, and passes them to the Decode stage, which decodes the syllables and fetches register operands with the decoded register identifiers. In essence, this stage decodes each syllable to an operation, access registers and performs control transfers (in the Branch Unit). The accessed registers are send as operands to the Execute 1 stage. This stage performs arithmetic and logic operations (in

the ALU and MUL units) on its operands. The Execute 2 stage performs a pre-selection of functional results for the commit stage and does the store or load phase of data memory operations in the load/store unit. The WriteBack stage performs all write activities and avoids read-after-write data hazards that are created by the decode stage. The register file write targets can be in the General Purpose Register file (GR) and/or in the Branch Register file (BR). The data memory of the processor is implemented by utilizing the Block RAM (BRAMs) that are provided by the FPGA.

2.3 Related Work

This section presents the prominent related work for design of low-power, area-efficient and high-throughput video processing units in H.264/AVC. It covers both the ASIC as well as FPGA based solutions. In-loop deblocking filter designs are presented in Section 2.3.1. The related work for intra-prediction unit in H.264/AVC is listed in Section 2.3.2, followed by the state-of-the-art presented in the literature for forward (inverse) integer transform units in H.264/AVC in Section 2.3.3.

2.3.1 In-loop Deblocking Filter

From a low-power deblocking filter hardware design point of view, a number of architectures and their hardware implementations have been presented in literature for last few years [24, 26–56]. For instance, in [44, 45], the authors presented two different low-power hardware implementations for the deblocking filter. Their design consists of a 2-stage pipeline datapath. The first pipeline stage includes a 12-bit adder and two shifters. The second stage includes a 12-bit comparator, several two's complement units and multiplexers for conditional branching. The dynamic power consumption of the deblocking filter is significantly reduced with such a simplified datapath. However, because of lack of sufficient storage registers, the intermediate results are to be stored and fetched from the memory several times during filtering processing. This results in a large number of cycles to process a single MB (5248 cycles / MB). Therefore, the throughput of the proposed design is reduced quite significantly. Moreover, no attempts were made, or at least reported to remove the redundancy in the deblocking filter algorithm. The hardware implementation of proposed design works at 72 MHz in a Xilinx Virtex II FPGA and can process up to Common Intermediate Format (CIF, 352×288) video frame

resolution only.

Similarly, Byung-Joo, et al., also proposed a low-power deblocking filter in [36]. They attempt to reduce the dynamic power consumption by skipping filtering process completely or partially by exploiting the relationship between block level encoding parameters and filtering conditions. The proposed design, however, requires 2272 cycles to process one MB and, therefore, can only process up to CCIR601 video frame format (704×576) in real-time. The proposed design though provides better throughput, but still do not meet the real-time processing requirement of a wide range of portable multimedia devices where standard-definition (SD, 720×480), high-definition (HD, 1280×720) or even higher resolutions are to be supported.

Recently, Nam Thang, et al., [43] also proposed a low-power, high-throughput 4-stage pipelined architecture, implemented using 0.18 μm CMOS standard cell technology, for deblocking filter in H.264/AVC. Four stages pipeline implementation of deblocking filter and hybrid processing order enable them to process a single macroblock in 192 cycles. While working at 220 MHz, it can provide a throughput up to 1146 K MB/s. The filter core consumes 34.8 μW for QCIF frame resolution at an operating frequency of 220 MHz. The proposed design uses five 4×4 block buffers for intermediate result storage during MB processing to achieve high throughput and attempts to reduce power consumption by adopting clock gating for these blocks. However, no such attempts are made for 2nd and 3rd pipeline stages where most of the filter processing take place. Moreover, no partial or full filtering process skipping scenarios are taken care off in the proposed architecture to reduce the dynamic power.

From real-time video processing applications point of view, a large number of deblock filter accelerators using single filter unit have been presented in the literature. For instance, in [46], Shih, et al., propose a 5-stage pipelined hardware architecture for deblocking filter. They employ a novel filtering order and data reuse strategy to reduce the number of cycles, memory traffic and required area for their implementation. The authors of [32] re-arrange the data flow to significantly reduce the memory size requirement and propose in-place architecture which re-uses the intermediate data as soon as it is available and thus are able to reduce the intermediate data storage to four 4×4 blocks instead of a complete 16×16 macroblock . Similarly Chang in [31] also re-orders the computing flow to efficiently use the intermediate data between the adjacent edges in their proposed architecture.

The hardware architecture in [52] implements a parallel-in parallel-out recon-

figurable FIR filter to carry out the filtering operations and uses a dual port SRAM for intermediate data storage. Li, et al., [38] adopt a 2 dimensional parallel memory scheme for parallel access in both the horizontal and vertical directions to speed up the filtering process and also eliminate the need of a transpose circuitry by using this memory scheme efficiently.

A hybrid filter scheduling described in [40] reduces required number of clock cycles for filtering and thus improves the system throughput while using the column-of-pixel data arrangement to facilitate the memory accesses and reusing the pixel value. A 5-stage pipelined architecture proposed in [29] for simultaneous processing of strong and weak filtering modes uses a novel transpose design to reduce the hardware cost and an alternate processing order of vertical and horizontal edges to reduce the on-chip memory requirement.

The area requirement for some of these hardware accelerators [38] [31] [56], is significantly low. This reduction in area requirement is because of their reduced functionality as these architectures do not implement boundary strength (BS) computation module.

Some of the hardware solutions based on multiple filter units are also introduced in the literature quite recently. These solutions provide higher throughput at the cost of additional on-chip area, but still most of these solutions do not meet the throughput requirements of all the levels (level 1-5.2) offered by the video Ccodec H.264/AVC. For instance, F. Tobajas based on a double-filter strategy, and using a raster scan filtering order, proposes a hardware architecture in [48]. Cheng in [27] [28]proposes a configurable window based architecture to simultaneously filter in both the directions. The main idea is to reduce the number of memory references through simultaneous processing architecture (SPA) using the vertical processing order instead of the raster scan order. A similar architecture is proposed in [53] by Venkatraman, et al.

Some efforts are also made to optimize the filter kernels by removing redundant operations, however the strong filter mode is the main focus in most of these attempts. For instance, in [49], the author suggest 3 different decompositions of the filter kernels in Strong filter mode to reduce the number of addition operations. The author however, do not consider similar decompositions for Weak Filter mode.

In short, single filter unit based solutions require less area in terms of equivalent gate count fort their implementation but fail to meet the processing requirements of high definition video in real-time. In some of the cases area reduction is achieved through reduced functionality offered by these hardware designs. The solutions based on multiple filtering units, on the other hand,

though provide better throughput at the cost of additional area but still do not meet the real-time processing requirement of all the levels offered by the video coding standard H.264/AVC.

2.3.2 Intra Prediction

Many different variants of intra-prediction algorithm has been presented in literature for last few years [47, 57–73]. The recent research on H.264/AVC intra prediction has focused on efficient hardware designs. The work presented in [71] proposes a hardware design with 5-stage registers and three configurable data paths for intra prediction unit. It uses reconfigurable predictor generators that exploit inherent parallelism within one prediction mode. The proposed design can process approximately 100 VGA frames in real-time. The proposed design, process each prediction mode individually and therefore, does not exploit the full space of optimization and parallelism between the different intra prediction modes. Since the proposal made in this work processes the intra prediction modes sequentially. Consequently, it requires a relatively higher frequency to achieve a throughput of 100 VGA frames in real-time. Moreover, it also suffers from overhead of configuring the hardware for different prediction modes.

In [68], an efficient intra-frame codec is proposed. The proposed solution can process $720p$ @ 30 fps in real-time and can be used for both the encoder and decoder implementations. This solution, however, excludes the plane mode for 16×16 luma and 8×8 chroma blocks. The proposed solution, therefore, can be used in a matched encoder-decoder scenarios only where it is guaranteed that plane mode is not used for intra-prediction.

High-throughput hardware designs for a H.264/AVC decoder are proposed in [47] and [65]. The proposed designs can process high-definition (HD) video in real-time. Since no attempts were made to optimize the intra prediction algorithm to reduce the arithmetic operations, therefore, the hardware implementation of these designs cost significant amount of hardware resources (approximately 29K gates).

Similarly, in [72], an efficient hardware implementation for intra-prediction unit is proposed. The design targets the H.264/AVC encoder and uses a so-called combined module approach to generate a subset of intra-prediction modes in parallel. Furthermore, the intra-prediction algorithm is optimized to significantly reduce the number of arithmetic operation for the computation of prediction samples. The proposed solution, however, does not fully eliminate

the redundancy in the algorithm and also primarily targets the H.264 encoder.

2.3.3 Integer Transform

A number of efficient algorithms and hardware designs for forward and inverse-integer transform in H.264/AVC, are presented in literature in recent past [22,64,74–94].From the perspective of hardware designs for integer transform in H.264/AVC, the main focus of research has been to develop fast algorithms with reduced on-chip area for its hardware implementation. In [82], Liu et al., used a parallel register array to realize the transpose operation for the 2D 4×4 forward integer transform. In [89], Cheng proposed high-throughput 4×4 integer transform architecture with no transpose memory requirement. Fan, et al., proposed a fast 2D transform algorithm in [79]. Similarly in [88,93,95], a number of fast algorithms for the integer transform have been proposed.

The H.264/AVC encoder utilizes multiple transforms (Forward/Inverse) integer transform and (Forward/Inverse) Hadamard Transform. Therefore, recently, the research focus has been shifted to design a unified transform unit to support multiple transforms in H.264/AVC [80], [94], [90], [92]. The motivation for such efforts is to reduce the on-chip area for the hardware implementation of these transforms units by sharing area between them. The proposed architectures although support multiple transforms in H.264/AVC, but most of them provide only one type of transform at a time. All of these architectures take residual data as an input and provide the transformed coefficients for the selected type of transform at the output.

2.4 Summary

In this chapter, we have presented an overview of the latest and state-of-the-art video coding standard H.264/AVC. Different functional units of the video codec are briefly described and the compute-intensive units are identified. Afterward, ρ-VEX Reconfigurable VLIW processor is introduced. The processor's architecture and design is also explained. Finally, the related work for the identified compute-intensive video coding functional units in H.264/AVC is presented in this chapter.

3

Deblocking Filter

THE adaptive deblocking filter in H.264/AVC not only provides perceptually improved video quality by removing blocking artifacts in the reconstructed video frames, but also helps to reduce bit-rate typically between 5% -10% [2]. However, this improvement in video quality and reduction in video bit-rate is achieved at the cost of increased complexity of deblocking filter algorithm.

The complexity of deblocking filter algorithm in H.264/AVC is mainly based on high adaptivity of the filter, which requires conditional processing on block edge and sample levels. Another reason for high complexity is small block size employed for residual coding in H.264/AVC coding algorithms. With 4×4 blocks and a typical filter length of 2-samples in each direction, almost every sample in a video frame must be loaded from memory, either to be modified or to determine if neighboring samples will be modified.

According to analysis of run-time profile of H.264/AVC decoder sub-functions, deblocking filter consumes about one-third of the total computational resources required for video decoder [96]. This is true even though the deblocking filter can be implemented without any multiplication or division operations. These demanding characteristics suggest a high-throughput implementation for such a filter, especially for High Definition (HD) video applications such as Digital TV (DTV) , where even larger frame-size at higher frame-rate is to be processed in real-time. Similarly a low-power solution is required for portable battery-powered multimedia electronic devices, such as mobile phones, digital cameras, and Personal Digital Assistant (PDA).

The main contributions of this chapter are:

1. For image processing applications, a single-filter-unit based low-power hardware design for deblocking filter in H.264/AVC.

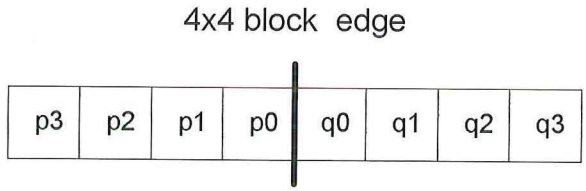


Figure 3.1: Convention for describing pixels across vertical and horizontal edges in a macroblock.

2. For real-time video processing applications, a high-throughput, area-efficient hardware design for deblocking filter in H.264/AVC using dual filter units.

This chapter is organized as follows: Section 3.1 briefly describes the algorithm for deblocking filter in H.264/AVC. The design considerations for deblocking filter-core are presented in Section 3.2. In Sections 3.3 and 3.4, we propose two hardware designs for deblocking filter, targeting image processing applications on battery-powered electronic devices and real-time video processing applications respectively. Finally, Section 3.5 summarizes the chapter.

3.1 Deblocking Filter Algorithm

The convention to describe pixels across vertical and horizontal edges in a macroblock is depicted in Figure 3.1. The bold line between pixels p0 and q0 is either a vertical or horizontal edge between two adjacent 4×4 blocks. Pixels q0-q3 represent pixels in the current 4×4 block whereas pixels p0-p3 are from corresponding left or top neighbor 4×4 block across vertical or horizontal edge respectively.

The filtering operation is performed on macroblock (MB) basis after reconstruction of the video frame, with all MBs in a video frame processed in order of increasing MB address. The filtering is applied to all 4×4 block edges except edges at the boundary of a picture or for which the filtering is disabled explicitly. The filtering process is invoked for luma and chroma components of a MB separately. For each MB and for each component, vertical edges are filtered first starting with the left most edge and proceeding through the edges in their geometrical order. The horizontal edges are filtered afterward in a similar fashion, starting with the top most edge and proceeding through the edges in their geometrical order [7] as depicted in Figure 3.2. The filtering process

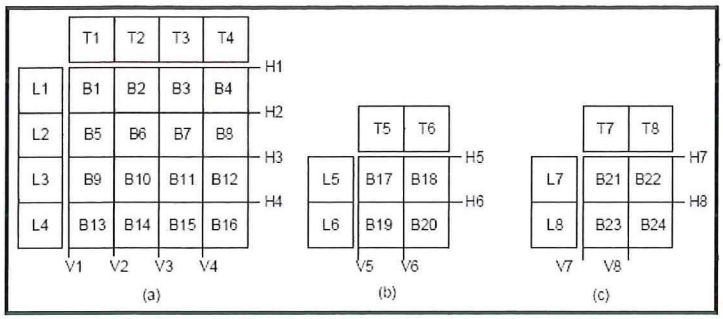


Figure 3.2: Vertical and horizontal 4×4 block edges in a macroblock. (a) Luminance (Y) component of a macroblock. (b) Chrominance (Cb) component of a macroblock and (c) Chrominance (Cr) component of a macroblock.

also requires pixels from left and top neighbor macroblocks in order to filter the left most edges (V1, V5 and V7) and the top most edges (H1, H5 and H7) respectively. A macroblock at 4×4 block level is shown in Figure 3.2. Blocks B1 through B16 belong to luma (Y) component of the current MB. Similarly, B17-B20 and B21-B24 are from chroma (Cb, Cr) components of the same MB respectively. Blocks (T1-T8) are top neighbor 4×4 blocks and (L1-L8) represent left neighbor 4×4 blocks of a MB in Figure 3.2.

The H.264/AVC deblocking filter is a highly adaptive filter. It adapts at slice level, 4×4 block-edge level, and set of pixels level within a 4×4 block. At slice level, a set of threshold parameters control filter operation, while at block-edge level, filter strength is computed on the basis of parameters such as, encoding mode and motion vector difference. The boundary strength (Bs) parameter controls filter strength at 4×4 block level and varies between 4 (strong filter case) and 0 (no filter case). For Bs values 1, 2 and 3, a weak filtering process is invoked on pixels across 4×4 block edge. The decision process to compute Bs value is described in [7]. The selected filter mode is turned ON or OFF depending upon the value of Filter Sample Flag (FSF). The FSF value is derived from equation on line 2 in Algorithm 3.1.

In strong filter mode (Bs = 4), at most 3 pixels are modified on either side of the edge. New values for pixels on left or top of the edge (p-pixels) are computed using equations on lines 10-12, Algorithm 3.1, provided strong filter flag for p-pixels (SFF_P) is set. If SFF_P is not set, only one pixel (p0) is filtered and the new value for this pixel is derived by equation provided on line 14, Algorithm 3.1. Similarly, pixels in the current block (q-pixels) are computed using equations on lines 17-19, Algorithm 3.1, provided the corresponding

Algorithm 3.1 : Deblocking filter algorithm for single block edge.

Require: Pixels p_0 - p_3 and q_0- q_3, $\alpha(Q_p)$, $\beta(Q_p)$, Bs,$ChromaEdge$,C_0

1: **begin**
2: $FSF := (Bs \neq 0)\ \&\ (|p_0 - q_0| < \alpha)\ \&\ (|p_1 - p_0| < \beta)\ \&\ (|q_1 - p_0| < \beta)$
3: $SFF\_P := Not(ChromaEdge)\&(|p_2 - p_0| < \beta)\&(|p_0 - q_0| < (\alpha/4+2))$
4: $SFF\_Q := Not(ChromaEdge)\&(|q_2 - q_0| < \beta)\&(|p_0 - q_0| < (\alpha/4+2))$
5: $WFF\_P := Not(ChromaEdge)\ \&\ (|p_2 - p_0| < \beta)$
6: $WFF\_Q := Not(ChromaEdge)\ \&\ (|q_2 - q_0| < \beta)$
7: **if** $FSF \neq 0$ **then**
8: **if** $Bs == 4$ **then**
9: **if** $SFF\_P \neq 0$ **then**
10: $p_0' := (p_2 + 2 * p_1 + 2 * p_0 + 2 * q_0 + q_1 + 4) \gg 3$
11: $p_1' := (p_2 + p_1 + p_0 + q_0 + 2) \gg 2$
12: $p_2' := (2p_3 + 3 * p_2 + p_1 + p_0 + q_0 + 4) \gg 3$
13: **else**
14: $p_0' := (2 * p_1 + p_0 + q_1 + 2) \gg 2$
15: **end if**
16: **if** $SFF\_Q \neq 0$ **then**
17: $q_0' := (q_2 + 2 * q_1 + 2 * q_0 + 2 * p_0 + p_1 + 4) \gg 3$
18: $q_1' := (q_2 + q_1 + q_0 + p_0 + 2) \gg 2$
19: $q_2' := (2q_3 + 3 * q_2 + q_1 + q_0 + p_0 + 4) \gg 3$
20: **else**
21: $q_0' := (2 * q_1 + q_0 + p_1 + 2) \gg 2$
22: **end if**
23: **else**
24: $\triangle := clip(c_1, -c_1, (((q_0 - p_0) \ll 2 + (p_1 - q_1) + 4) \gg 3))$
25: $p_0' := clip(0, 255, p_0 + \triangle)$
26: $q_0' := clip(0, 255, q_0 - \triangle)$
27: **if** $WFF\_P \neq 0$ **then**
28: $p_1' := p_1 + clip(c_0, -c_0, (p_2 + ((p_0 + q_0 + 1) \gg 1) - 2 * p_1) \gg 1)$
29: **end if**
30: **if** $WFF\_Q \neq 0$ **then**
31: $q_1' := q_1 + clip(c_0, -c_0, (q_2 + ((p_0 + q_0 + 1) \gg 1) - 2 * q_1) \gg 1)$
32: **end if**
33: **end if**
34: **end if**
35: **end**

strong filter flag (SFF_Q) is set. In case SFF_Q is not set, only one pixel (q0) is filtered using equation provided on line 21 Algorithm 3.1 and rest of the pixels remain unchanged. The values for SFF_P and SFF_Q flags are derived from equations described on lines 2 and 3, in Algorithm 3.1 respectively.

In weak filter mode (Bs = 1, 2 or 3), at most 2 pixels on either side of the edge are filtered. The new values for pixels p0 and q0 across the edge are derived from equations on lines 25 and 26 in Algorithm 3.1 respectively. Similarly, equations on lines 28 and 31 in Algorithm 3.1 are used to compute filtered values for pixels p1 and q1, provided the corresponding weak filter flag (WFF_P, WFF_Q) is set. In case weak filter flag is not set, filtering operation for these pixels is turned off.. The values for WFF_P and WFF_Q are derived from equations on lines 5 and 6, Algorithm 3.1. No filtering is applied for Bs = 0.

3.2 Design Consideration for Deblocking Filter Core

In this section, we elaborate on the optimizations carried out at deblocking filter algorithm level. Other design-level considerations along with their justification and/or motivation are also described, for low-power and area-efficient hardware design for core filtering unit in deblocking filter.

3.2.1 Processing Units for Luma and Chroma Components

The H.264/AVC supports YUV 4:2:0, YUV 4:2:2, and YUV 4:4:4 video frame formats. In case of YUV 4:2:0, 33% of pixels belong to chroma component of a macroblock. This number increases to 50%, and 66% for YUV 4:2:2 and YUV 4:4:4 video frame formats respectively. Since the chroma filtering is much cheaper than the luma filtering in terms of required arithmetic operations (13 vs. 45), it is potentially beneficial to design separate processing units for luma and chroma components of a macroblock. These separate luma-, chroma-processing units, depending upon the status of ChromaEdge flag (Algorithm 3.1), can be enabled or disabled independent of each other to reduce the dynamic power consumption.

3.2.2 Processing Units for Strong and Weak Filter Modes

For both luma and chroma components of a MB, the filter mode for pixels across any edge is determined on the basis of Bs value for that edge. Since

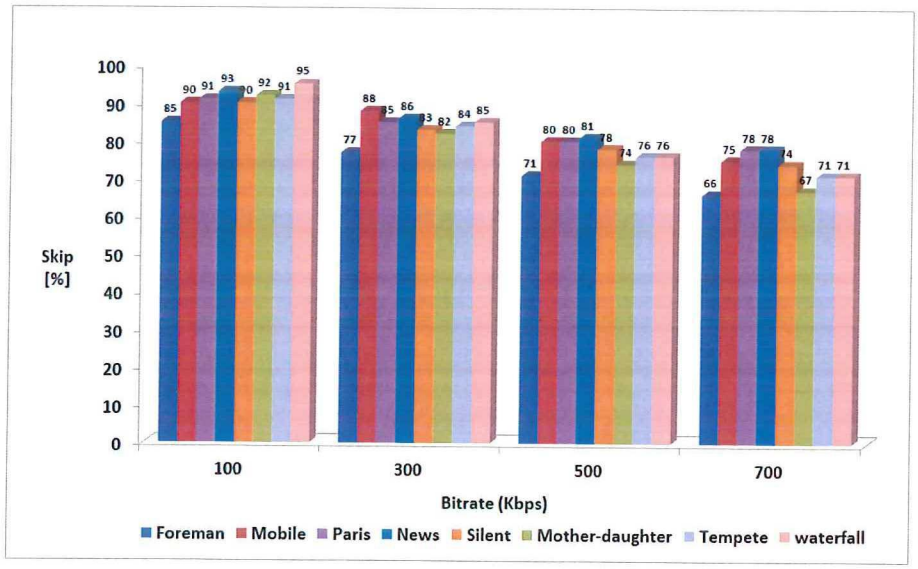


Figure 3.3: Full filtering process skip (Bs = 0 case).

only one mode is active at any given time during filtering process, therefore, the arithmetic operations for both filter modes are implemented in separate processing units employing clock gating. Depending on the filter mode, either of these processing units (strong or weak filtering units) is deactivated dynamically to potentially reduce the power consumption.

3.2.3 Full or Partial Filtering Process

The filtering process, for an edge with Bs = 0, is skipped. Moreover, the FSF is computed using pixel values across horizontal or vertical edge, and the corresponding 4×4 block level filter control threshold values. If FSF is not set, the filtering process for current pixels-row or pixels-column (vertical or horizontal filtering cases) within the 4×4 block can also be skipped.

For a variety of video test sequences encoded at different bit-rates (100 – 700 Kbps), Figure 3.3 illustrates the percentage cases when complete filtering process can be skipped because of Bs being zero. There is no need to even compute FSF and other conditional flags for such a 4×4 block case. Similarly, Figure 3.4 suggests that even when Bs is non-zero, there are significant number of cases when FSF is false and, therefore, filtering process can be skipped partially for current pixels-row or pixels-column within the 4×4 block.

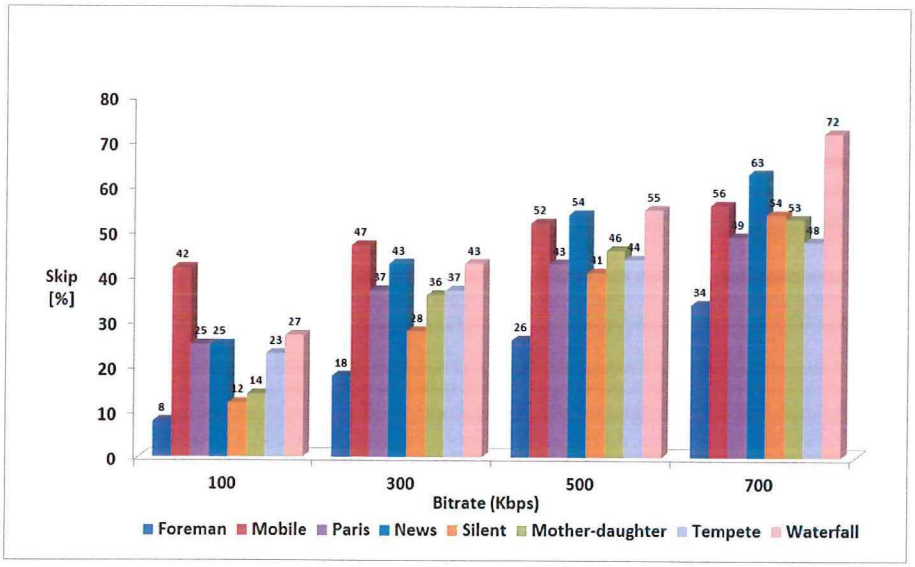


Figure 3.4: Partial filtering process skip (Bs = 1, 2, 3 or 4).

We propose to compute FSF along with other conditional filter flags in first pipeline stage as independent processing unit with gated clock. This unit is activated only if Bs is non-zero. With this proposal, the filtering process for the whole 4×4 block is skipped for a scenario when boundary strength is zero. In case Bs is not zero, the FSF along with other conditional flags are used to generate appropriate control signals for rest of the processing units in the 2^{nd} and 3^{rd} pipeline stage of deblocking filter, to partially skip filtering process and, therefore, reduce dynamic power.

3.2.4 Processing Units for Conditional Filtering

For conditional filtering scenario, we computed percentage of cases when any of the conditional filter flags (WFF_P, WFF_Q, SFF_P, SFF_Q) for all filter modes (strong and weak filter modes) is false. The data is generated for various video test sequences encoded at 100 – 700 Kbps video bit-rate. As depicted in Figures 3.5 and 3.6, there are quite a significant number of cases during processing of any video frame when the conditional filtering is off. Therefore, we propose to split the processing for these cases and implement them in separate units supported by clock gating. This helps to deactivate these processing units independently and, therefore, reduce dynamic power consumption when any

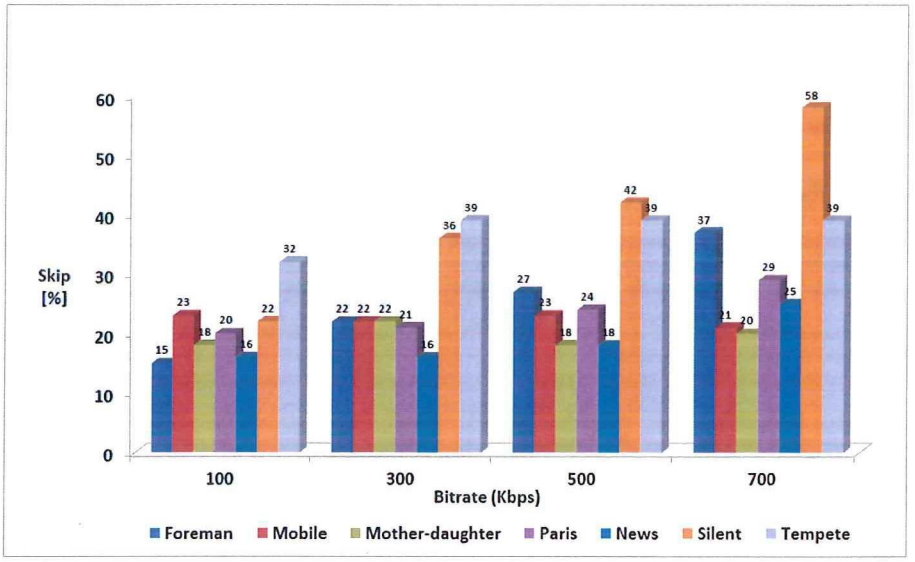


Figure 3.5: Conditional filtering process skip (Bs = 4).

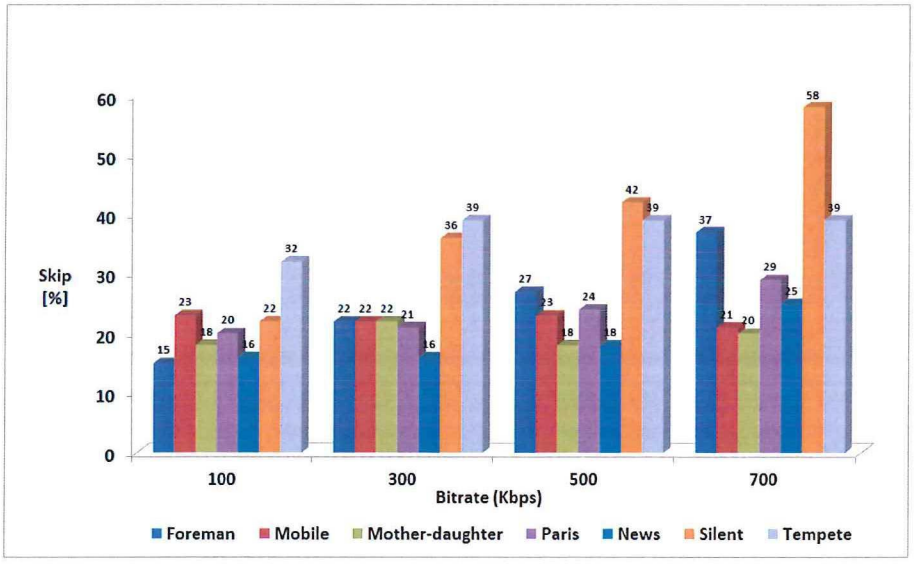


Figure 3.6: Conditional filtering process skip (Bs = 1, 2, or 3).

of the conditional filtering flags is not set.

3.2.5 Algorithm-level Optimizations for Deblocking Filter

In strong filter mode, filtered pixel values are derived from filter equations on provided lines 10-21 as provided in Algorithm3.1. Similarly, filtered pixel values for weak filter mode are derived from equations provided on lines 24-31 in Algorithm3.1. It is clear from these equations that strong filter mode requires 36 additions; whereas, 13 additions and 5 clip operations are required for weak filter mode. This results in 49 additions and 5 clip operations in total, while excluding the operations needed to compute the pixel level filter flags. As mentioned in the related work, the author in [49] proposed 3 different decompositions of the deblocking filter equations in strong filter mode to reduce the number of operations. However, no such attempt is made to reduce the number of operations for weak filter mode. The proposed decomposition in [49], with least number of operations, requires 22 addition operations for strong filter mode. Another 13 additions and 5 clip operations are, therefore, required for weak filter mode. This results in 35 additions and 5 clips operations to perform complete deblocking filtering process for an edge, in all filter modes. Similarly, a deblock filter design with 23 additions for strong filter mode is presented in [97] . The proposed design, therefore, requires 36 additions and 5 clip operations for its hardware implementation for both deblock filter modes in H.264/AVC.

3.2.5.1 Decomposition of Filter Kernels

One of the important contributions of this work, is an attempt to remove the redundant operations in filter equations for both filtering modes (Algorithm 3.1) through novel decomposition of filter equations as depicted in Algorithm 3.2 . Note that the rounding constants in original filter equations on lines 10-21 in Algorithm 3.1 are efficiently distributed among several intermediate results in the form of x + y + 1. This operation, however, requires only one adder for its hardware implementation. The term 4 * u0 + 3 on line 25 in Algorithm 3.2 can be realized by appending bits '11' as least significant bits in u0. , and therefore, do not require any additional adder. Similarly, reuse of intermediate results in optimized algorithm helps to significantly reduces addition operations. Our proposed decomposition requires only 31 additions.

Algorithm 3.2 : Optimized deblocking filter algorithm for single block edge.

Require: Pixels p_0 - p_3 and q_0- q_3, $\alpha(Q_p)$, $\beta(Q_p)$, Bs,$ChromaEdge$,C_0

1: **begin**
2: $u_1 := (2 * p_2 + p_0 + 1) + (q_0 - 4 * p_1)$ when $Bs \neq 4$
3: $u_1 := (p_2 + p_0 + 1) + (q_0 + p_1 + 1)$ when $Bs = 4$

4: $u_2 := (2 * q_2 + q_0 + 1) + (p_0 - 4 * q_1)$ when $Bs \neq 4$
5: $u_2 := (q_2 + q_0 + 1) + (p_0 + q_1 + 1)$ when $Bs = 4$

6: $u_3 := (p_1 - q_1)$ when $Bs \neq 4$
7: $u_3 := (p_1 + q_1 + 1)$ when $Bs = 4$

8: $u_4 := p_0 + q_0 + 1$
9: $u_5 := u_3 + u_4$

10: $t_1 := p_3 + p_2 + 1$
11: $t_2 := q_3 + q_2 + 1$
12: $t_3 := p_1 + p_0 + 1$
13: $t_4 := q_1 + q_0 + 1$

14: **if** $StrongFilter(Case : Bs = 4)$ **then**
15: $p_0' := (u_1 + u_5) \gg 3$
16: $p_1' := (u_1) \gg 2$
17: $p_2' := (u_1 + t_1) \gg 3$
18: $p_{0c}' := (u_3 + t_3) \gg 2$

19: $q_0' := (u_2 + u_5) \gg 3$
20: $q_1' := (u_2) \gg 2$
21: $q_2' := (u_2 + t_2) \gg 3$
22: $q_{0c}' := (u_3 + t_4) \gg 2$
23: **end if**
24: **if** $WeakFilter(Case : Bs = 1, 2or3)$ **then**
25: $\triangle := clip(c_1, -c_1, ((4 * u_0 + 3) + u_3 + 1) \gg 3)$
26: $p_0' := clip(0, 255, p_0 + \triangle)$
27: $q_0' := clip(0, 255, q_0 - \triangle)$

28: $p_1' := p_1 + clip(c_0, -c_0, (u_1) \gg 2)$
29: $q_1' := q_1 + clip(c_0, -c_0, (u_2) \gg 2)$
30: **end if**
31: **end**

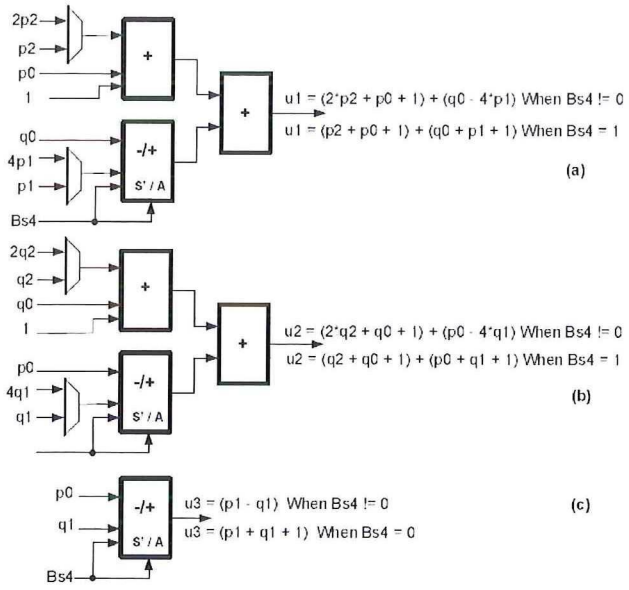


Figure 3.7: Overlapped data paths for u1, u2 and u3 in strong and weak filter modes.

3.2.5.2 Inter Filter Mode Optimizations

Since both deblocking filter modes are mutually exclusive, and therefore, only one of them is used to filter pixels data across the 4×4 block edge at any given time. Therefore, inter-filter-mode redundancy can be removed by designing datapath (for u1, u2, and u3) in such a way that they overlap as much as possible in terms of arguments for processing units (adders in this case). The proposed realization of filter equations for u1, u2 and u3, using overlapped datapath, is illustrated in Figure 3.7(a), 3.7(b), and 3.7(c) respectively. The inter-filter-mode optimization reduces another 7 adders at the cost of 4 multiplexers. The proposed decomposition of filter equations along with inter-filter-mode optimization requires only 24 additions and 5 clip operations for its hardware implementation. The number of additions are, therefore, reduced by 51% when compared with that of original deblocking filter equations in [7], by 31%, when compared with decomposition proposed in [49], and by 33%, when compared with [97]. A comparison of number of operations required to carry out deblocking filter process, proposed by [7], [49], and [97] and our proposal, is depicted in Figure 3.8.

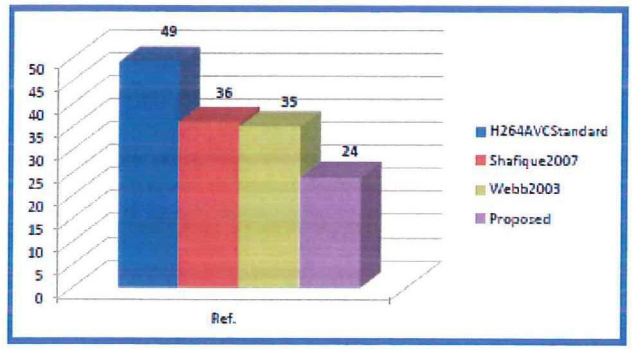


Figure 3.8: Comparison: Addition operations in strong and weak filter modes.

3.3 Low-power Deblocking Filter Design

In this section, we first introduce the top-level hardware design for deblocking filter in H.264/AVC. Subsequently, the internal design of the deblocking filter core unit is presented in this section. The independent processing blocks proposed in Section 3.2 for low-power hardware design are also identified and explained in the design of deblocking filter core unit.

3.3.1 High Level Organization

The block diagram of the hardware deblocking filter is depicted in Figure 3.9. The design is based on a single filter unit (filter core), and two RAM units (filter control data RAM, and neighbor 4×4 block data RAM). Since the filter process is identical in both horizontal and vertical directions, therefore, a single filter unit is used in the hardware design. During filtering process for vertical or horizontal block edge, pixel data from left or top 4×4 neighbor block, across the block edge, is also required. This left or top neighbor 4×4 block pixels data is stored in neighbor 4×4 block data RAM. Similarly, filter-control-threshold parameters (α, β, Tc_0, and Bs) at 4×4 block level are stored in filter control data RAM unit. The neighbor 4×4 block data RAM unit is composed of 32×32-bit dual port SRAM. Similarly, filter control data RAM is composed of 16×32-bit SRAM. All the datapaths in Figure 3.9 are 32-bit wide.

During deblocking filter process, the 4×4 block level filter control and neighbor pixel data is first transferred and stored in the corresponding RAM units. Later, the filtering process starts as soon as first pixel-row of 4×4 block B1

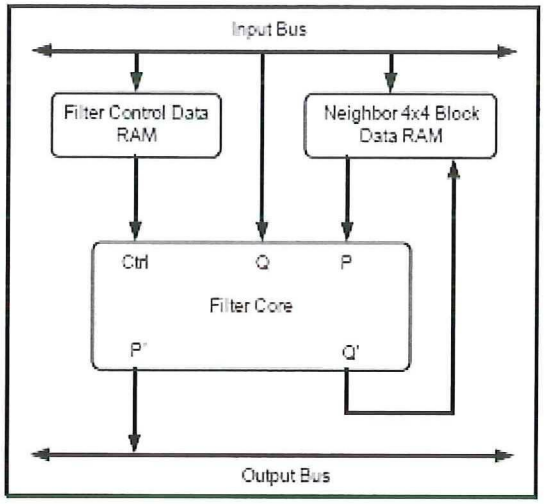


Figure 3.9: High level organization of the deblocking filter hardware implementation.

is received from the external memory at Q input (Figure. 3.9) of deblock filter core via input bus. The corresponding neighbor pixel data across the edge (P data input in Figure 3.9), is provided to the filter core from neighbor 4×4 block data RAM unit. Partially filtered macroblock data (Q′) of current 4×4 block is temporarily stored in the neighbor 4×4 block data RAM unit. This partially filtered stored data is used as neighbor pixel data for the next 4×4 block. Whereas, the filtered pixel data at the output P' is transferred to the external frame memory via output bus.

3.3.2 Deblocking Filter Core Unit

The deblock filter core is the main processing block where all filter related conditional flags are computed and the appropriate filter kernels are used to filter the input pixel data. The design choices made for this unit plays an important role in determining the throughput, power consumption and on-chip area for its implementation.

Pipeline processing is an effective implementation choice to achieve high throughput. In order to meet the real-time processing requirement of up to full-HD (1920×1088) resolution video frame format, while keeping the operating frequency as low as possible with low-power design point of view, we implement the deblock filter core in pipeline fashion. The straightforward pipeline implementation of deblocking filter, however, potentially consumes

high dynamic power because of parallel processing and unnecessary activity in different pipeline stages. We, therefore, propose to implement deblocking filter core using optimized algorithm and recommended design options in Sections 3.2.

The 1st pipeline stage of deblock filter core implements processing for evaluation of all the filter control flags (FSF, WFF_P, WFF_Q, SFF_P, and SFF_Q). These flags are used in filter control block pipeline stage 1, to generate appropriate control signals for filter processing blocks in subsequent pipeline stages of deblock filter core.

The 2nd and 3rd pipeline stages of filter core consist of processing blocks with gated clock, as identified in Section 3.2. ChromaBs4Block in Figure 3.10, as the name suggests, implements filter processing for chroma component of the macroblock for Strong Filter Mode (lines 14 and 21 in Algorithm 3.1). The deblocking filter process for pixels p0 and q0 in weak filter mode, is identical for chroma-, luma-components of a MB. Therefore, for this case, ChromaLumaBs123Block implements all the arithmetic processing for pixels p0 and q0 as described on lines 24-26 in Algorithm 3.1. Similarly, the overlapped datapath for conditional filtering in strong and weak filter modes is implemented in LumaCommonPBlock and LumaCommonQBlock, whereas, the LumaBs4_PBlock, LumaBs4_QBlock implement rest of the processing for strong filter mode case [16].

In case of strong or weak filter modes for chroma component of a MB, one can see from Figure 3.10, either ChromaBs4Block or ChromaLumaBs123Block is to be enabled respectively. While, all the processing blocks in 2nd and 3rd pipeline stages are deactivated, in this case, for 33% of the processing time in a MB. Similarly, for other filtering scenarios, filter control block in 1st pipeline stage generates appropriate control signals to activate only the respective processing block(s) in the 2nd and 3rd pipeline stages. If Bs is zero, all processing blocks are deactivated and the unfiltered pixels are transferred at the output of deblock filter core via by-pass stage (not depicted in Figure 3.10). The experimental results suggest that our hardware design for deblocking filter consumes significantly less dynamic power, when compared with state-of-the-art low-power deblocking filter designs presented in the literature.

3.4 High-throughput Deblocking Filter Design

In this section, for real-time video processing applications, we propose a high-throughput and area-efficient hardware design for deblocking filter. The pro-

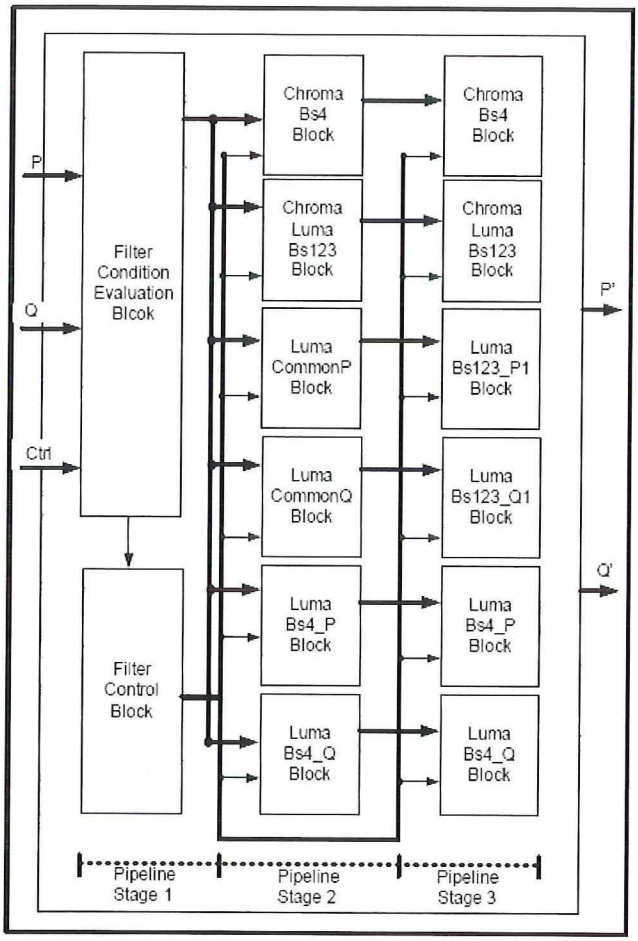


Figure 3.10: Block diagram of the DBF core.

posed design is based on dual identical filter cores and, therefore, process horizontal and vertical edges simultaneously. To achieve area-efficiency and reduce dynamic power consumption, the filter core is implemented using optimized decomposed filter kernels , already discussed in the previous sections. In this section, different building blocks that constitute top-level design of hardware deblocking filter accelerator are first introduced with a brief rationale for each of them. Subsequently, the pixel data flow, at 4×4 block level, is explained to provide an insight in the functioning of the proposed hardware accelerator.

3.4.1 High Level Organization

The block diagram of deblocking filter hardware accelerator is depicted in Figure 3.11. All solid line datapaths are 32-bit wide (4 pixels) and the dotted lines represent control signals. This accelerator is based on two identical filter units vertical edge filter (VEF), horizontal edge filter (HEF), two transpose units (Trans1 and Trans2), boundary strength computation unit (BS), and two left/top 4×4 neighbor blocks RAM units (LFNB RAM, TPNB RAM). VEF unit processes pixel-rows across vertical block edge in horizontal direction. Similarly, HEF unit processes pixel-columns across horizontal block edge in vertical direction so as to remove the blocking artifacts in a MB. The LFNB RAM stores pixel-rows of left neighbor 4×4 blocks (L1-L8) for vertical edge filtering process in VEF. Similarly, TPNB RAM stores pixel-columns of top neighbor 4×4 blocks (T1-T8) for horizontal edge filtering process in HEF.

During the filtering process of the internal block edges, partially filtered pixels from current 4×4 block are used as left and top neighbor pixels for the next 4×4 block along horizontal and vertical direction respectively. Therefore, LFNB/TPNB RAM units also serve as a temporary storage to hold partially filtered intermediate pixel results.

The BS unit computes boundary strength for all 4×4 block edges in a MB in both horizontal and vertical directions. The required configuration data, e.g., encoding type, motion vectors and quantization parameters of 4×4 blocks, for computation of boundary strengths, are stored in RAM unit as depicted in Figure 3.11. The RAM unit is composed of two 8×(4×8-bit) dual-port SRAMs and each one of these is used to store either top or left neighbor block configuration data during boundary strength computation process. The BS unit provides these computed boundary strength values along with filter control thresholds (α, β, and Tc_0), stored in the ROM table, to VEF and HEF filter units during filtering process.

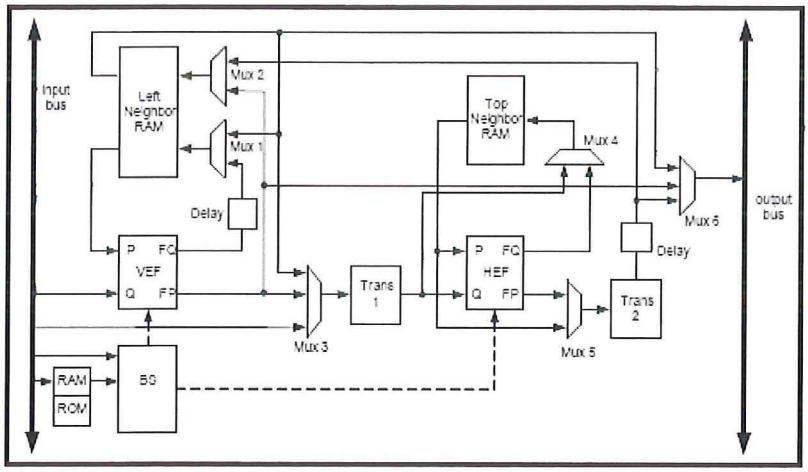


Figure 3.11: High level organization of the proposed deblock filter accelerator.

The Trans1 unit, at the input of HEF unit, transposes incoming pixel-rows of 4×4 block into pixel-columns. The other transpose unit i.e. Trans2, at the output of the HEF unit, converts the filtered pixel-columns back to pixel-rows before transferring them to external picture buffer via output bus.

3.4.2 Data Flow in Deblocking Filter Hardware Accelerator

Before starting filtering process for any MB, the configuration data and the top neighbor 4×4 blocks (T1-T8) are transferred from external memory to deblocking filter hardware accelerator. During this phase, BS unit computes boundary strength for all block edges in current MB. The pixel data for luma component in current MB follows configuration data and is transferred in raster scan order on 4×4 block level (B1, B2, ..., B16). Deblock filter phase starts as soon as first pixel-row of block B1 arrives at the input of VEF filter unit. The input pixel data is first filtered for vertical block edges in VEF unit and later by HEF unit for block edges in horizontal direction. The chroma 4×4 blocks (B17, B18, ..., B24) follow luma blocks and are filtered in the same fashion.

Once the filtering operation is completed in both directions by VEF and HEF units for the luma and chroma blocks, these filtered blocks are transferred to external picture buffer via output bus. The data-flow at block level is depicted in Figure 3.12. The VEF(Q-input) is always provided with input blocks (B1, B2, ..., B24) from external unfiltered picture buffer. The corresponding left neighbors (L1, B1, B2, B3, L2, ...) are fed from LFNB RAM unit into VEF(P-

input) filter unit.

The partially filtered blocks (B1, B2, ..., B24) from VEF (Q-output) are always stored in LFNB RAM and are used as left neighbor blocks in next filter block cycle. The filtered blocks (B1, B2, ..., B23), after completion of vertical edge filtering operation from VEF unit, are transferred to next filtering unit HEF for horizontal edge filtering operation. The last 4×4 block in each block row of current MB (B4, B8, B12) are temporarily stored in the LFNB RAM, and are also sent to HEF unit via the same data path before start of next block row. HEF unit also receives 4×4 blocks of current MB, after being filtered in horizontal direction by VEF unit, in the same sequence as that of VEF unit (Q-input), as shown in Figure 3.12.

The last column of 4×4 blocks (B4, B8, B12, B16, B18, B20, B22 and B24) in current MB are stored in LFNB RAM as (L1-L8) for next macroblock after completion of filter operation in both directions. The blocks in the last luma, chroma block rows (B13, B14, B15, B17, B19, B21 and B23), temporarily stored in the TPNB RAM module are sent to the external picture buffer. These blocks are transposed from pixel-columns to pixel-rows orientation during the transfer process by Trans2 unit. The left neighbor blocks of the previous MB are directly sent from LFNB RAM.

When vertical filtering process for current MB is completed, transfer of configuration data and top neighbor blocks (T1-T8) in next MB is initiated, in parallel with the processing of HEF unit for current MB, to maximize the throughput of the deblock filter.

3.4.3 Transpose Unit in Deblocking Filter Hardware Accelerator

Our deblocking filter hardware accelerator utilizes two identical filter units to achieve high throughput. However, because of identical filter units, two transpose units are required in the design as depicted in Figure 3.11. These transpose units are an overhead of such an arrangement and cost significant amount of area for their implementation. Different techniques have been presented in the literature to reduce area cost of transpose units. All of these techniques, in one way or another, require a separate temporal register file for its realization.

We, however, follow a different approach in our design to minimize this overhead. We identify that the transpose process and the boundary strength computation process are two mutually exclusive set of operations in time. The storage used for BS parameters is a free resource during the transpose phase. Therefore, in our design we re-use RAM units s as a storage during the transpose

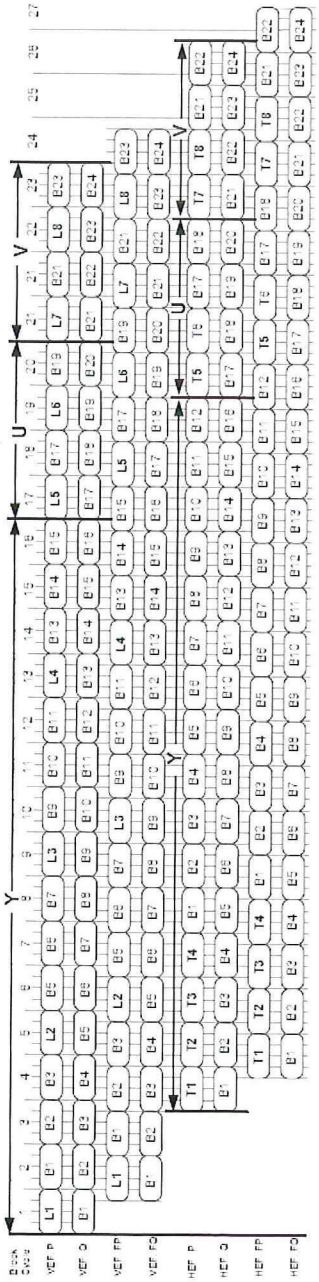


Figure 3.12: 4×4 block level data flow through the VEF, HEF units in the proposed hardware architecture.

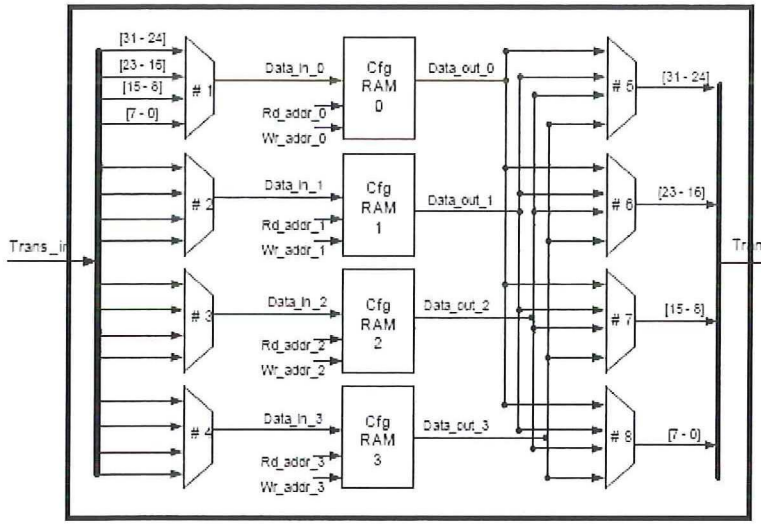


Figure 3.13: Functional block diagram of Transpose unit.

phase. This saves us precious area when compared to the use of dedicated register files, during the transpose phase. The transpose units in our design do not require any separate temporal buffer for their implementation. However, it do requires some pre- and post-storage pixel data re-arrangements for implementation of transpose units. The design and implementation details of transpose units using configuration RAM units are explained in the remainder of this section.

3.4.3.1 Transpose Unit Implementation

The transpose unit consists of 2 sets of 4 multiplexers connected to RAM units as depicted in Figure 3.13. RAM units are shared between transpose and BS units to serve as temporary storage location. The cost of a register file consisting of 14, 32-bit registers [48] is about 3.3k gates whereas the multiplexers depicted in Figure 3.13 cost only 500 gates (implemented on 0.18μm CMOS standard cell technology). This optimization delivers area saving. The transpose mechanism is explained in the following:

The transpose unit takes 8 cycles to complete transpose process of a 4×4 block. In first 4 cycles, block "n" is stored at "set 1" address locations (0, 1, 2 and 3). While writing into the RAM unit, the pixel-rows of block "n" are arranged in such a way that no two pixels of any pixel-column of transposed

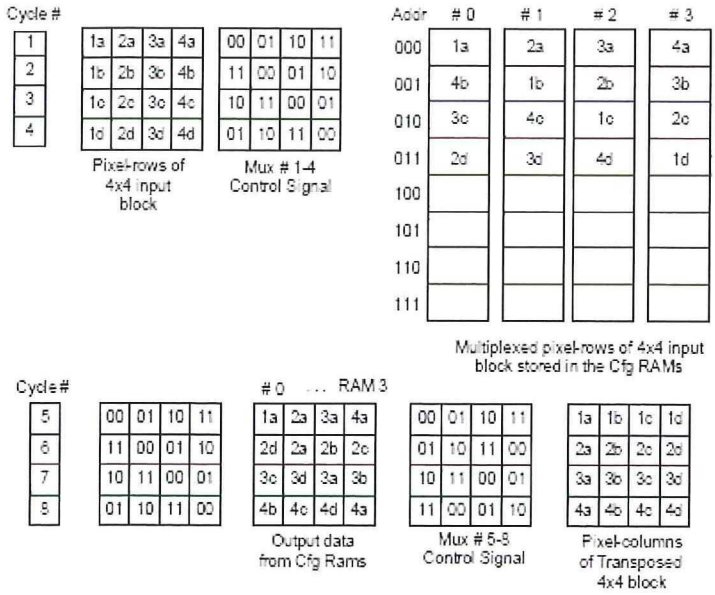


Figure 3.14: Transpose mechanism for 4×4 block.

4×4 block occupy same storage RAM unit. This is achieved through multiplexers Mux1-Mux4 at the input of RAMs as shown in Figure 3.13. The control signals for these multiplexers and data stored in RAM units are depicted in Figure 3.14(a). In next 4 cycles, block "n+1" is stored after same pixel rearrangement process at "set 2" address locations (4, 5, 6 and 7). Meanwhile, pixels from previously stored block "n" at "set 1" address locations are read out from RAMs using appropriate addresses, as depicted in Figure 3.14(b).

These pixels are subsequently re-positioned to form pixel-columns of 4×4-transposed block using multiplexers Mux5-Mux8 at the output of RAM units. Similarly, in the next 4 cycles, block "n+2" is stored at "set 1" address locations while block "n+1" is read out from "set 2" address locations and so on. Figure 3.14(a) depicts the control signal for Mux1-Mux4 during cycles c1 to c4 along with re-arranged data stored in the RAM units. Read addresses and the corresponding data read from RAM units is depicted in Figure 3.14(b). Similarly, the control signals for multiplexers Mux5-Mux8 with and the output of transpose unit are also depicted in Figure 3.14(b). During BS computation phase, multiplexers Mux1-Mux8 controls are set such that no re-positioning of the pixel data is done at both ends of the RAM units.

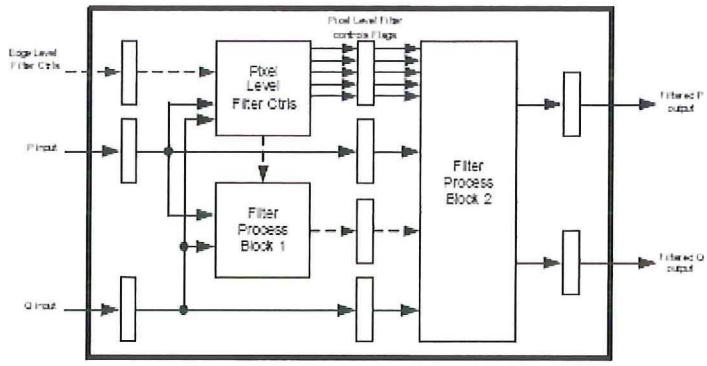


Figure 3.15: Functional blocks for vertical and horizontal edge filter units.

3.4.4 Vertical and Horizontal Filter Units

This section describes the design of VEF and HEF units and related optimizations employed at this level. The block diagram of this unit is depicted in Figure 3.15. Since all filtering operations are carried out in these units, therefore, more than two third of the total area of the deblocking filter hardware accelerator is occupied by these units. The design of this unit plays an important role in determining the throughput and overall area cost for the deblocking filter hardware accelerator.

3.4.4.1 Efficient Pipeline Design

The intermediate results storage registers between pipeline stages cost a significant amount of area. To minimize this overhead, we implement common datapath between two deblocking filter modes (u1, u2 and u3 in Algorithm 3.2) in the first pipeline stage. This results in only 6 intermediate results (u1, u2 and u3) for both filter modes. This number is further reduced to 3 intermediate results due to the fact that both filter modes (strong and weak filter modes) are mutually exclusive as depicted in Figure 3.7(a), 3.7(b), and 3.7(c). From pixel level filter control flags perspective, we designed the pipeline stages such that all processing to compute pixel level filter controls is also in first pipeline stage. This design choice requires only 5 one-bit flags to be passed on to the subsequent filter pipeline stages. This design choice, however, results in a longer processing chain for computation of filter control flags and is on the critical path in our design. Therefore, the maximum operating frequency for

our hardware accelerator is determined by the choice of implementation of pixel level filter control block. Implementation level optimization for critical path, explained below, enabled us to achieve 166 *MHz* operating frequency.

3.4.4.2 Critical Path Optimization

The computation of filter control flags is on the critical path for our deblock filter hardware design. The equations to derive the values for filter control flags are provided in Algorithm 3.1. The common operations in computation of these flags can be written as follows:

Flag A = Abs(x – y) < Threshold ? 1 : 0.

where x and y represent the pixel values.

The straightforward implementation is depicted in Figure 3.16(a) and consists of following operations: Compute difference (x - y). If the computed difference is negative, take 2's complement to produce absolute difference and finally determine the flag A by comparing the absolute difference with threshold value. This choice of implementation is composed of three sequential operations. One of the possible modification is to compute two difference results (x-y) and (y-x) in parallel, as depicted in (Figure 3.16(b)), and then select the positive difference as an absolute difference using a multiplexer, for the subsequent comparison operation. This implementation shall help to achieve a higher clock frequency in comparison to the previous implementation choice, as the multiplex operation is more cost effective than sign change operation in terms of speed, but at the cost of additional subtraction operations.

In our implementation, we removed the absolute difference computation stage and merged it into the comparison stage as depicted in Figure 3.16(c). The sign bit of difference result determines the Add/Sub control. Subsequently, the sign bit of the result of Add/Sub component determines the status of the flag. This results in a further reduction in the sequential operations without any additional operation in parallel with the first stage and, therefore, helps to achieve even higher clock frequency (166 MHz). One requirement for such an implementation is that now we need to provide threshold value that is one less than the actual value. Since these threshold values are loaded from the ROM table and are only used for the computation of filter control flags, therefore, we can either store these modified threshold values in the ROM table or this operation of modification can be implemented in BS unit. In either case, the threshold modification is not on the critical path anymore, resulting in less sequential operations and, therefore, a higher clock frequency.

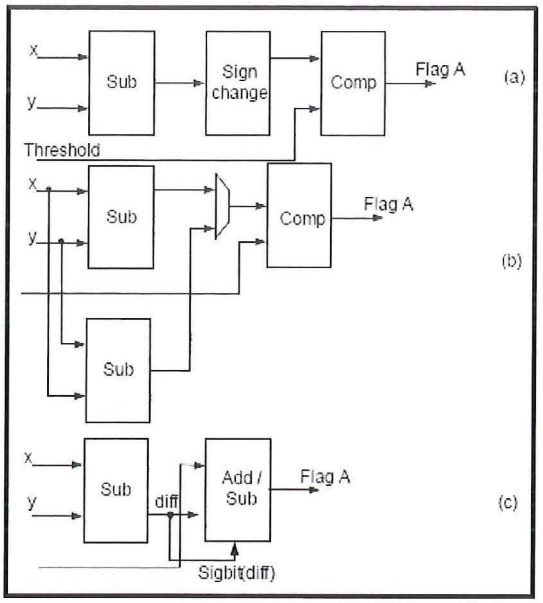


Figure 3.16: Implementation of basic building block in pixel level filter control block.

3.5 Summary

In this chapter, we have presented a low-power hardware design for deblocking filter in H.264/AVC. we also proposed a novel decomposition of deblocking filter kernels to reduce number of operations by more than 51%. The algorithmic optimizations significantly reduces the area cost for its hardware implementation. A number of independent processing units are identified in the optimized deblocking filter algorithm. These independent processing units, when implemented using clock gating, helps to reduce dynamic power consumption of deblocking filter.

For real-time video processing applications, a high-throughput and area-efficient hardware deblocking filter is also presented in this chapter.. The proposed design for deblocking filter in H.264/AVC provides real-time filtering operation for HDTV video format (4096×2304, 16:9) at 30 fps and also meet throughput requirements of all levels (levels 1-5.1) in H.264/AVC video coding standard.

Note.

The content of this chapter is based on the the following papers:

Muhammad Nadeem, Stephan Wong, Georgi Kuzmanov and Ahsan Shabbir, *A High-throughput, Area-efficient Hardware Accelerator for Adaptive Deblocking Filter in H.264/AVC*, proceedings of International Symposium on Embedded Systems For Real-time Multimedia 2009 (ESTIMEDIA), pp. 18-27, Grenoble, France, October 2009.

Muhammad Nadeem, Stephan Wong, Georgi Kuzmanov, Ahsan Shabbir, M.F. Nadeem and Fakhar Anjam, *Low-power, High-throughput Deblocking Filter for H.264/AVC*, proceedings of International Symposium on System-on-Chip 2010 (SoC), pp. 93-98, Tampere, Finland, September 2010.

4

Intra Prediction

THE H.264/AVC video coding standard supports multiple directional intra prediction modes to reduce the spatial redundancy in the video signal. There 4 intra prediction modes for luma 16×16 block type, 4 intra prediction modes for chroma 8×8 block type, and 9 intra prediction modes for 4×4 block type in H.264/AVC. These multiple intra prediction modes help to significantly improve the encoding performance of intra-frame encoder. Studies have shown that H.264/AVC outperforms JPEG2000, a state-of-the-art still-image coding standard, in terms of subjective as well as objective image quality [66]. This makes H.264/AVC intra-frame codec an attractive choice to be utilized as an image compression engine. Applications like Digital Still Camera (DSC) employ intra-frame encoding technique to compress high-resolution images.

In a video frame encoded in intra mode, the current macroblock is predicted from previously encoded neighboring macroblocks from the same video frame. A video frame with all MBs encoded in intra mode does not depend on any other video frame(s) and, therefore, can be decoded independently. Consequently, a video encoded with intra-frames only, is easier to edit than a video encoded with inter-frames (frames predicted from past or future video frames). Similarly, in many surveillance systems, video is compressed using intra-frames encoding mode due to legal reasons. Courts in many countries do not accept predicted frames as legal evidence [98]. As a result, a typical video surveillance system compresses videos using intra-encoded frames only. Consequently, intra-only video coding is widely used coding technique in television studio broadcast, digital cinema and surveillance video applications.

The H.264/AVC compliant baseline decoder is approximately 2.5 times more time-complex than H.263 compliant baseline-decoder [66]. According to analysis of run-time profiling data of H.264/AVC baseline decoder sub-functions, intra-prediction is one of the top 3 compute-intensive functions [66]. A high-

throughput intra-frame processing chain is, therefore, an important requirement for H.264/AVC intra-frame codec for real-time video processing applications. The demanding characteristics of intra-prediction algorithm suggest its hardware implementation for high-definition video applications, where even larger frame size at higher frame rates are to be processed in real-time.

In this chapter, we describe a high-throughput and area-efficient design for intra-prediction unit in H.264/AVC compliant video codecs. The main contributions are as follows:

1. A proposal to optimize the intra-prediction algorithm for 4×4 luma blocks by decomposing the filter kernels. The proposed decomposition significantly reduces additions operations for its hardware implementation (27% - 60% reduction).
2. An efficient hardware design of intra-prediction unit to reduce on-chip area by using hardware sharing approach for mutually exclusive processing scenarios (approximately 21K gates).

Furthermore, we also optimize the common equations in intra-prediction algorithm, to efficiently compute intra-predicted samples for 16×16 luma and 8×8 chroma blocks.

The chapter is organized as follows. Section 4.1 provides an overview of intra prediction algorithm and also describes proposed optimizations in the algorithm. The proposed configurable hardware design is presented in Section 4.2, whereas, Section 4.3 summarizes this chapter.

4.1 Intra Prediction Algorithm

In this section, we briefly introduce intra-frame processing chain in H.264/AVC video decoder. Subsequently, an overview of intra prediction algorithm for 4×4 luma blocks, 16×16 luma blocks, and 8×8 chroma blocks is provided in separate subsections. The optimizations to reduce number of operations in intra prediction algorithm are also explained along with the algorithm overview.

The functional block diagram for an H.264/AVC intra-frame decoder is depicted in Figure 4.1. The entropy decoder unit parses input bitstream and decodes intra-prediction mode used for current MB. This intra-prediction mode information is then passed on to intra prediction unit to generate the intra-predicted pixels block as per given intra-prediction mode.

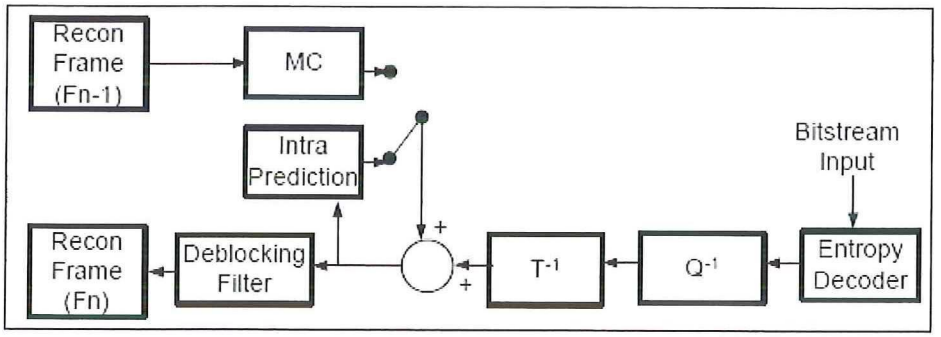


Figure 4.1: Functional block diagram of H.264/AVC decoder.

The residual block can be computed in a parallel datapath using inverse quantization (Q^{-1}) and the inverse integer transform (T^{-1}) processing units as depicted in Figure 4.1. Once intra prediction block is made available by intra prediction processing unit, the current pixel block is reconstructed by adding the residual block to the predicted block. The unfiltered reconstructed pixels from current block are used as neighbor or boundary pixels for computation of predicted samples for the next block in the video frame.

The H.264/AVC video coding standard supports multiple intra-prediction modes for 4×4 and 16×16 luma pixels block , and 8×8 chroma pixels block, as explained in the following subsections.

4.1.1 Intra Prediction for 4×4 Luma Blocks

There are 9 intra-prediction modes supported for 4×4 luma-pixels block in H.264/AVC. Each of these prediction modes generates a 4×4 predicted pixel block using some or all of the neighboring pixels as depicted in Figure 4.2. Since the intra predicted samples for Vertical (VT) and Horizontal (HZ) prediction modes are same as the boundary pixels, therefore, these intra-prediction modes do not require any processing and are easy to implement. Similarly, DC-prediction mode computes an average value using valid boundary pixels as predicted pixels block. The remaining 6 intra-prediction modes (modes 3 to 8), however, compute predicted samples using 2- or 3-taps filter kernels. The filter equations for these modes in H.264/AVC video coding standard require 59 addition operations [7] for the computation of prediction samples. Efforts have been made to reduce these addition operations by efficient reuse of intermediate results. The optimized datapath proposed in [72] requires 33 addition

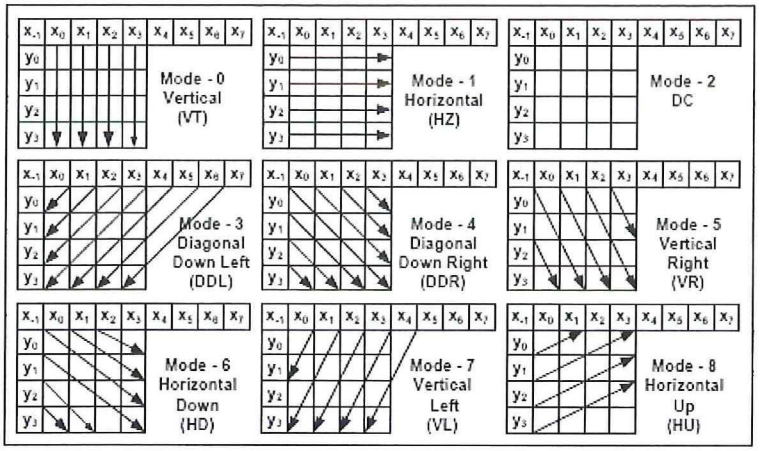


Figure 4.2: Illustration of nine 4×4 luminance prediction modes.

operations and is the lowest number reported in literature.

We propose to decompose filter kernels for luma intra 4×4 modes 3 to 8. The optimized intra prediction filter equations after decomposition are depicted in Algorithm 4.1. With the proposed decomposition, the number of additions to compute intra predicted samples for these modes are reduced to 24. Our approach provides 60% reduction in addition operations when compared with unique intra-prediction equations in H.264/AVC video coding standard and 27% reduction when compared with a proposal in state-of-the-art [72].

4.1.2 Intra Prediction for16×16 Luma, and 8×8 Chroma Blocks

The H.264/AVC video coding standard supports 4 intra-prediction modes for 16×16 luma and 8×8 chroma blocks. The horizontal, vertical, and DC modes are similar to those for 4×4 luma blocks and are easy to implement. The fourth intra prediction mode, also known as plane intra-prediction mode, utilizes bi-linear function to compute intra-predicted samples. Since 8×8 chroma and 16×16 luma plane intra-prediction modes are similar, therefore, we only discuss that for 8×8 chroma blocks in this section. The algorithm for 8×8 chroma intra-plane mode is described in Figure 4.3.

The algorithm computes three variables a, b and c (Figure 4.3: lines 8, 9, and 10) using intermediate results H and V (Figure 4.3: lines 13, and 14). We optimize the computation of H and V, and propose another set of equations as described on lines 8 and 11 in Figure 4.4. The proposed equation to compute H

Algorithm 4.1 : Proposed decomposed filter kernels for intra-prediction modes.

Require: Pixels x_{-1} - x_7 and y_0- y_3

$s_1 = x_{-1} + x_0 + 1$
$s_2 = x_0 + x_1 + 1$
$s_3 = x_1 + x_2 + 1$
$s_4 = x_2 + x_3 + 1$
$s_5 = x_3 + x_4 + 1$
$s_6 = x_4 + x_5 + 1$
$s_7 = x_5 + x_6 + 1$
$s_8 = x_6 + x_7 + 1$
$s_9 = x_{-1} + y_0 + 1$
$s_{10} = y_0 + y_1 + 1$
$s_{11} = y_1 + y_2 + 1$
$s_{12} = y_2 + y_3 + 1$

$t_1 = s_1 + s_2 + 1$
$t_2 = s_2 + s_3 + 1$
$t_3 = s_3 + s_4 + 1$
$t_4 = s_4 + s_5 + 1$
$t_5 = s_5 + s_6 + 1$
$t_6 = s_6 + s_7 + 1$
$t_7 = s_7 + s_8 + 1$
$t_8 = s_8 + 2 * x_7 + 1$

$t_9 = s_9 + s_{10} + 1$
$t_{10} = s_{10} + s_{11} + 1$
$t_{11} = s_{11} + s_{12} + 1$
$t_{12} = s_{12} + 2 * y_3 + 1$

```
1. Intra Chroma Plane Prediction Mode-3
2. If (p[x, -1] and p[-1, y] with x = 0 .. 7 and, y = -1 ..7 are available) Then
3.
4.     The values of the prediction samples pred_c[x,y] are derived as follows:
5.     pred_c[x,y] = Clip (a + b * (x - 3) + c * (y - 3) + 16 >> 5)
6.
7.     Where
8.     a = 16 * (p[-1, 7] + p[7,-1])
9.     b = (5 * H + 32) >> 6
10.    c = (5 * V + 32) >> 6
11.
12.    and H and V are specified as
13.    H = Σ (x' +1) * (p[4+x', -1] - p[2-x', -1], where x' = 0 .. 3
14.    V = Σ (y' +1) * (p[-1, 4+y'] - p[-1, 2-y'], where y' = 0 .. 3
15.
16 . End If
```

Figure 4.3: Intra-prediction algorithm for 8×8 luminance block.

is partially overlapped with datapath for computation of 4×4 intra-prediction block. The intermediate results (t2, t6, s1 and s8) are computed using common overlapped datapath. Consequently, the datapath for computation of H (and also for V) is shared between plane intra-prediction mode for 16×16 luma, 8×8 chroma blocks and intra-prediction modes for 4×4 luma blocks. The original equations in the intra-prediction algorithm for computation of H and V requires 16 arithmetic operations (8 additions + 8 subtractions). It should be noted that we already replaced multiplications by shift and additions to come up this numbers. The proposed equations (Figure 4.4: line 8 and 11), because of shared datapath, require only 9 additional arithmetic operations for its implementation in hardware; instead of 16 arithmetic operations in original equations proposed in H.264/AVC video coding standard for computation of H and V. Similarly, pred[x,y] (Figure 4.3: line 5) is modified to extract the constants involved, and are computed once (Figure 4.4: variable k in line 16). Finally, these constants (k, b, and c) are used to derive the prediction samples of plane mode using Equation 4.1.

$$pred_c\,[x, y] = Clip\,[(A + B + C) \ll 5] \tag{4.1}$$

Where

$$A = \begin{pmatrix} k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k \\ k & k & k & k & k & k & k & k \end{pmatrix} \tag{4.2}$$

$$B = \begin{pmatrix} 0 & b & 2b & 3b & 4b & 5b & 6b & 7b \\ 0 & b & 2b & 3b & 4b & 5b & 6b & 7b \\ 0 & b & 2b & 3b & 4b & 5b & 6b & 7b \\ 0 & b & 2b & 3b & 4b & 5b & 6b & 7b \\ 0 & b & 2b & 3b & 4b & 5b & 6b & 7b \\ 0 & b & 2b & 3b & 4b & 5b & 6b & 7b \\ 0 & b & 2b & 3b & 4b & 5b & 6b & 7b \\ 0 & b & 2b & 3b & 4b & 5b & 6b & 7b \end{pmatrix} \tag{4.3}$$

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c & c & c & c & c & c & c & c \\ 2c & 2c & 2c & 2c & 2c & 2c & 2c & 2c \\ 3c & 3c & 3c & 3c & 3c & 3c & 3c & 3c \\ 4c & 4c & 4c & 4c & 4c & 4c & 4c & 4c \\ 5c & 5c & 5c & 5c & 5c & 5c & 5c & 5c \\ 6c & 6c & 6c & 6c & 6c & 6c & 6c & 6c \\ 7c & 7c & 7c & 7c & 7c & 7c & 7c & 7c \end{pmatrix} \tag{4.4}$$

4.2 Intra Prediction Unit Hardware Design

The top-level hardware design for intra prediction unit is presented in the section. Subsequently, data-flow in the core processing block of intra prediction unit, for generation of various intra-prediction modes, is described. The top-level hardware organization of the intra-prediction unit is depicted in Figure 4.5. It consists of neighbor pixel buffer block, prediction samples selection block, control block, and intra-prediction core processing block.

The neighbor pixel buffer, as the name suggests, is used to stores boundary pixels from the neighbor pixel-blocks. For a 4×4 pixels-block, depending

1. Optimized Intra Chroma Plane Prediction Mode-3
2.
3. $H = (x_4 - x_2) + 2 * (x_5 - x_1) + 3 * (x_6 - x_0) + 4 * (x_7 - x_{-1})$
4. $\Rightarrow H = (x_4 + 2 * x_5 + 3 * x_6 + 4 * x_7 + 4) - (x_2 + 2 * x_1 + 3 * x_0 + 4 * x_{-1} + 4)$
5. $\Rightarrow H = ((x_4 + 2 * x_5 + x_6 + 2) + 2 * (x_6 + x_7 + 1) + 2 * x_7)) -$
6. $((x_2 + 2 * x_1 + x_0 + 2) + 2 * (x_0 + x_{-1} + 1) + 2 * x_{-1}))$
7. $\Rightarrow H = (t_6 + 2 * s_6 + 2 * x_7) - (t_2 + 2 * s_1 + 2 * x_{-1})$
8. $\Rightarrow H = (t_6 - t_2) + 2 * ((s_3 - s_1) + (x_7 - x_{-1}))$
9.
10. Similarly
11. $V = (t_6 - t_2) + 2 * ((s_6 - s_1) + (x_7 - x_{-1}))$
12.
13. $pred_c[x,y] = Clip ((k + b*x + c*y) >> 5)$, From line 5
14. Where $k = a - 3*(b + c) + 16$
15. $\Rightarrow k = 16 * (x_7 + y_7) - 3*(b + c) + 16$, From line 8
16. $\Rightarrow k = ((x_7 + y_7 + 1) << 4) - ((b + c) + ((b + c) << 1))$

Figure 4.4: Optimized intra-prediction algorithm for 8×8 luminance block.

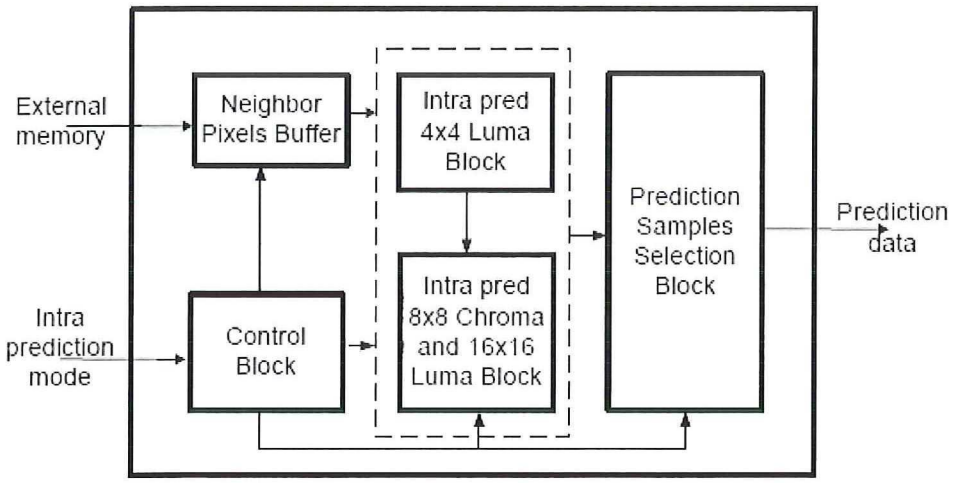


Figure 4.5: Top-level organization of intra-prediction unit in H.264/AVC Decoder.

upon the decoding order, block position, and encoding conditions, up to 13 boundary pixels (8-bit each) are stored in neighbor pixel buffer. Similarly, for 16×16 block, it provides storage registers to holds 33 boundary pixels at most. These boundary pixels are required in core intra prediction processing unit, for computation of prediction samples. The same buffer is also used to store partial results during computation of DC and plane intra-prediction modes, in case of 16×16 luma and 8×8, chroma blocks.

All processing to compute any given intra-prediction mode takes place in intra prediction core processing unit as depicted by dotted block in Figure 4.5. The detailed internal design of intra prediction core processing unit is depicted in Figure 4.6. The vertical and horizontal intra-prediction modes for all luma and chroma blocks (4×4, 8×8 and 16×16) do not require any computation. The boundary pixels for these modes bypass the intra-prediction core block (not depicted in Figure 4.6). In case of 4×4 luma intra-prediction modes 3 to 8, all unique prediction samples are generated in 1 clock cycle. Similarly, in case of 4×4 luma intra-prediction mode 2 (DC mode), the prediction sample (or DC value) is computed in 2 clock cycles. This is a worst-case scenario for luma 4×4 intra-prediction modes. The core processing unit computes unique sample values only. The distribution of these prediction samples to form a 4×4 prediction block is performed in the prediction samples selection block. In case of plane-mode, intra prediction core processing unit takes 6 clock cycles to computes constants (b, c and k) and the corresponding offset values for b and c (nb, nc). Using these constants, while computing 16 intra-predicted pixels in each cycle, all 64 intra-predicted pixels in 8×8 chroma predicted-pixels block are generated in subsequent 4 clock cycles. Therefore, it takes just 10 clock cycles in total to deliver the plane-mode, intra 8×8 chroma predicted-pixels block. Similarly 22 clock cycles are required to compute the plane-mode, intra 16×16 luma predicted-pixels MB.

In most of the 4×4 intra-prediction modes, some of the prediction samples are identical in the final intra-predicted pixels block. Figure 4.7 illustrates this fact for DDL and VL intra-prediction modes where some of the prediction samples (t_x and s_x in Figure 4.7; defined in Algorithm 4.1) are re-distributed in the predicted block. For such cases, prediction sample selection unit is responsible for this re-distribution of unique prediction samples in the final 4×4 predicted block accordingly, and transfer it to the next processing unit in the intra-processing chain of H.264/AVC decoder.

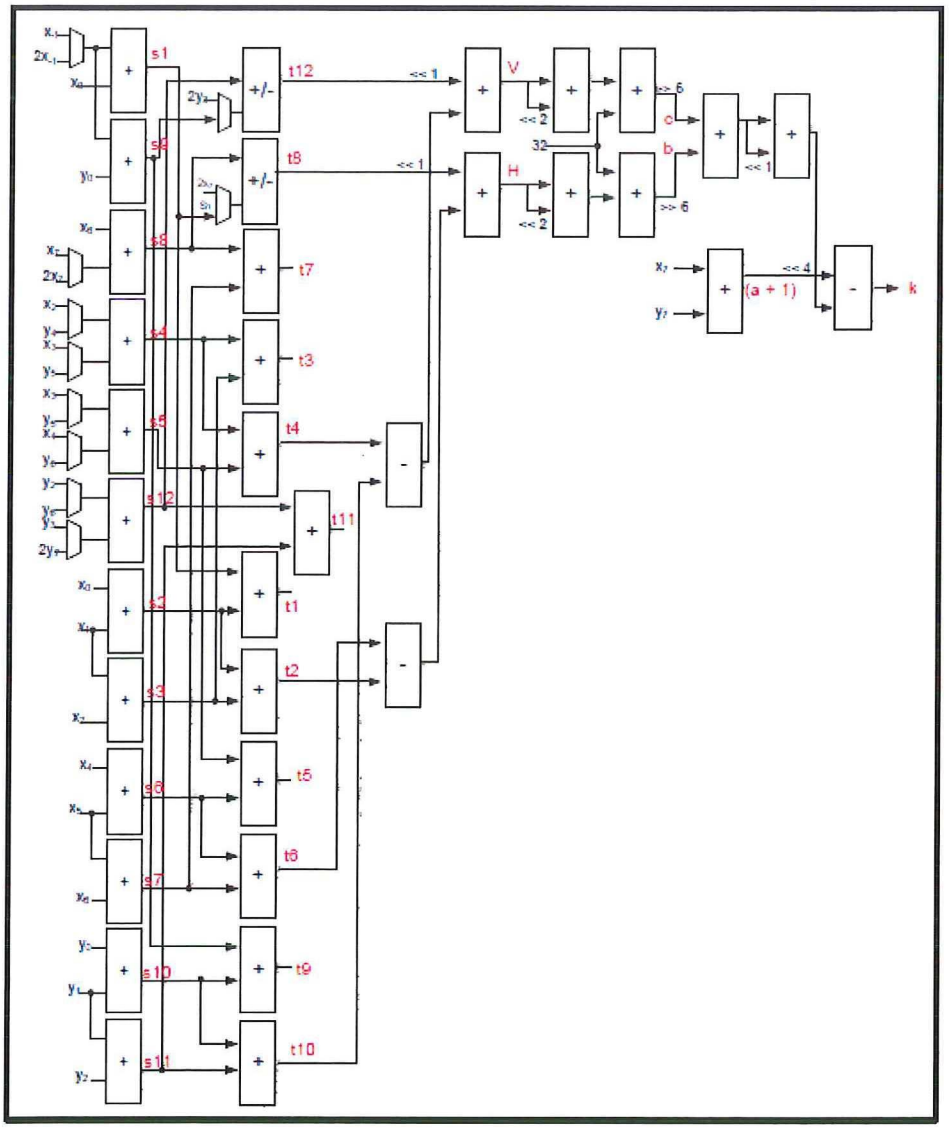


Figure 4.6: Intra-prediction core processing unit.

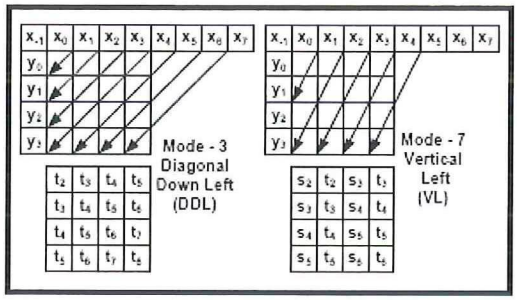


Figure 4.7: Prediction samples distribution in 4×4 DDL and VL modes.

4.3 Summary

In this chapter, we have proposed a high-throughput and area-efficient hardware design for the intra-prediction unit in the H.264/AVC. The optimization of algorithm for 4×4 intra prediction modes enabled us to reduce the number of additions by 60% when compared with those proposed in H.264/AVC video codec standard. Similarly, overlapped datapath for the 4×4 ,8×8, and 16×16 pixels-block also helps to greatly reduce area-cost for hardware implementation of intra prediction algorithm. With an operating frequency of 150 MHz, the proposed intra prediction unit can easily meet the real-time processing requirement of HDTV video frame formats.

Note.

The contents of this chapter are based on the the following paper:

Muhammad Nadeem, Stephan Wong, and Georgi Kuzmanov, *An Efficient Hardware Design for Intra-prediction in H.264/AVC Decoder*, proceedings of International Conference on Electronics, Communications and Photonics 2011 (SIECPC),pp. 1-6, Riyadh, Saudi Arabia, April 2011.

5

Forward Integer Transform

THE H.264/AVC, like its predecessors, is a hybrid codec. However, it is the first video coding standard that uses an integer inverse-transform design for its main spatial transform. The corresponding forward transform used in the encoder is, therefore, also an integer transform. The major advantage of this forward, inverse transform-pair being integer is that there is no possibility of a mismatch between standard-compliant encoder and decoder, The previous video coding standards such as H.261, H.262/MPEG-2, H.263, MPEG-1 and MPEG-4 part 2 all suffers encoder-decoder mismatch problem.

In a video encoder, forward integer transform is used to de-correlate video data at 4×4 predicted pixels-block level, preceding the quantization process. Beside integer transform, H.264/AVC standard also apply Hadamard transform to a block of DC-values in a macroblock for both luma and chroma components of video signal. The forward and inverse integer transform is computed for each 4×4 predicted block along with Hadamard transform for DC- blocks in a H.264/AVC video encoder. Additionally, the Hadamard transform is commonly used to compute Sum of Absolute Transformed Difference (SATD), as part of intra-mode decision process to choose best prediction block in inter-frame encoding process. The spatial transform processing in H.264/AVC codec is, therefore, one of the top 3 compute-intensive processing units in H.264/AVC codec.

In this chapter, we propose efficient hardware designs for realization of forward integer transform in H.264/AVC video encoder. The main contributions of this chapter are as follows:

1. A low-power, area-efficient hardware design to realize forward integer transform in H.264/AVC encoder. This design targets image compression applications, running on battery-powered electronic devices, such as Digital Still Cameras (DSC).

2. For real-time video compression applications, a high-throughput, low-latency, and area-efficient hardware design to realize forward integer transform in H.264/AVC encoder.

The chapter is organized as follows: Sections 5.1 and 5.2 describe the proposed solutions for image processing applications and real-time video processing applications respectively. Finally, Section 5.3 summarizes the chapter.

5.1 Forward Integer Transform for Image Processing Applications

In this section, we propose a solution to realize forward integer transform for image processing/compression applications using H.264/AVC codec. Since image processing applications are typically not real-time applications, therefore, compression-time or throughput of image compression engine is of less importance. However, as most of these applications run on battery-powered electronic devices, dynamic-power-consumption reduction is of prime importance for such applications.

Image compression applications compress the input image using intra-encoding technique. H.264/AVC video coding standard supports multiple directional intra-prediction modes to reduce spatial redundancy in the input signal. Putting it more precisely, H.264/AVC supports 4 intra-prediction modes at 16×16 pixels-block level and 9 intra-prediction modes at 4×4 pixels-block level, to significantly reduce the predictive error by exploiting spatial correlation between block to be encoded and its already encoded neighboring blocks in the input image or a frame. When a current pixles-block is to be encoded in intra-mode, a corresponding prediction block is formed using previously encoded and reconstructed pixels-blocks. Intra-prediction-mode-decision unit in an encoder, is responsible to determine the best intra-prediction mode for to be encoded current pixels-block at hand. The reference software for H.264/AVC encoder utilizes a Lagrange cost function to make an intra-prediction mode decision and the cost for an intra-prediction mode is computed as follows:

$$J_{Mode}(MB_k, I_k|Qp, \lambda_{Mode}) = D(MB_k, I_k|Qp) + \lambda_{Mode} * R(MB_k, I_k|Qp, \lambda_{Mode}) \quad (5.1)$$

Where λ_{Mode} , D and R represent Lagrange parameter, distortion and rate, respectively. The rate (R) is estimated by number of bits to encode the mode,

while, distortion (D) is measured using Sum of Absolute Difference (SAD) or Sum of Absolute Transformed Differences (SATD) for residual pixels-block data. SATD, however, is more precise in estimating the distortion and, therefore, helps to provide better image quality. The H.264/AVC reference software utilizes Hadamard transform to compute frequency domain coefficients of a residual block for candidate intra-prediction mode [99]. The distortion D in Equation. 5.1 is estimated by using the absolute sum of all Hadamard transformed coefficients.

We follow a different approach from related works cited in Section 2.3.3. As mentioned earlier, Hadamard-transformed residual block is used to estimate distortion (D) for the candidate intra-prediction mode J_{Mode} in an intra-prediction-mode-decision unit [99]. We propose to use same Hadamard-transformed residual block of the selected intra-prediction mode and derive the forward-integer-transformed coefficients for residual block in H.264/AVC compliant encoder. This approach requires less than half the number of additions (30 vs. 64) when compared with those for state-of-the-art designs in literature.

5.1.1 S-Transform: Hadamard-transformed Coefficients to Integer-transformed Coefficients Conversion

H.264/AVC utilizes an integer transform (C) to convert spacial domain 4×4 residual pixel block (X) into frequency domain (Z) as follows: .

$$Z = C \cdot X \cdot C^T \tag{5.2}$$

where

$$C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \tag{5.3}$$

Similarly, 2-D Hadamard transformed coefficients (Y) for 4×4 input pixels-block (X) s are computed using the Hadamard transform (H) as follows:

$$Y = H \cdot X \cdot H^T \tag{5.4}$$

where

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \tag{5.5}$$

If S is such a transform that it can be used to convert the Hadamard transformed coefficients (Y) into integer transformed coefficients (Z) Z , then for 1-D case, we can write as follows:

$$S = C \cdot H^{-1} \tag{5.6}$$

where S is provided as follows:

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & -0.5 & 0 & 1.5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0.5 \end{pmatrix} \tag{5.7}$$

5.1.2 Signal-flow for 1-D S-Transform

The proposed 1-D, S-transform in Equation 5.7 takes the Hadamard transformed coefficients and derive the integer transform coefficients. Only 4 additions and 2 shift operations are required for the implementation of 1-D, S-transform. While, on the other hand, a fast 1-D forward integer transform algorithm presented in [22] requires 8 additions along with 2 shift operations to compute the same integer-transformed results from intra-predicted residual block. The signal flow diagram for 1-D, S-transform is depicted in Figure 5.1.

5.1.3 2-D, S-Transform Algorithm

For 1-D S-transform computation case, only two Hadamard-transformed coefficients are required to be processed for any given column or row in Y, as depicted in Figure5.1(a). A 2-D integer transform can be realized using two 1-D S-transform, depicted in Figure 5.2. This approach, however, has an overhead as it requires a transpose unit between two successive 1-D transforms.

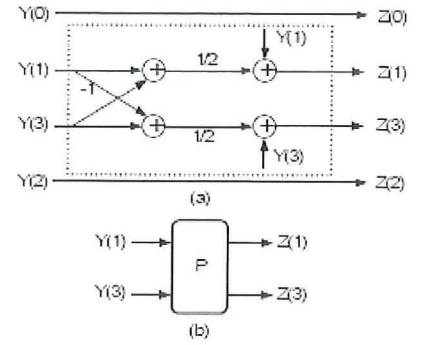


Figure 5.1: (a) Signal flow diagram for 1-D S-transform, (b) Basic processing block for 1-D S-transform.

Figure 5.2: Row-column processing for 2-D transform.

Another approach to realize 2-D integer transform, without any transpose unit, using row-column processing, is depicted in 5.3. This organization requires a total of 32 additions and 16 shift operations to generate integer-transformed coefficients from Hadamard-transformed coefficients data. In contrast, if the forward integer transform coefficients are generated from the intra prediction residual data, the fast algorithm using row-column computation method in [95] and [80] requires 64 additions and 16 shifts operations. Therefore, our proposal reduces the number of additions by half, when compared with the current state-of-the-art method in literature. The hardware implementation for this approach costs 32 adders only. The required number of adders are further reduced by removing redundant operations between cascaded P processing units in Figure 5.3 using direct 2-D optimized transform algorithm .

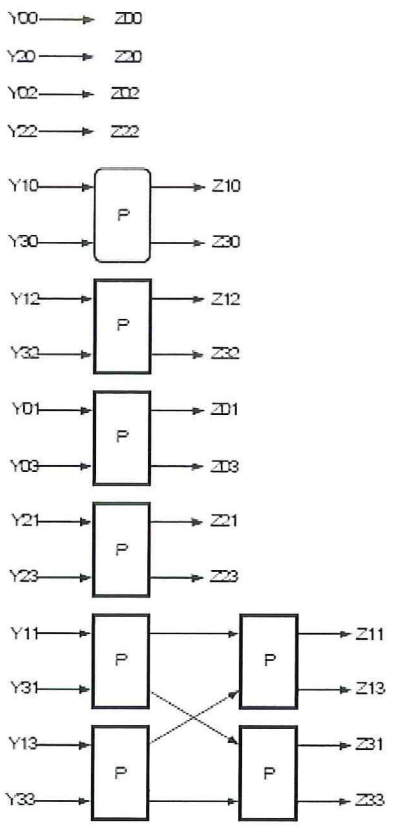


Figure 5.3: Functional block diagram for 2-D transform.

5.1.4 Optimized Algorithm for 2-D S-Transform

Direct 2-D algorithm for S-transform in Equation 5.7, can be derived as follows:

$$vec(Z) = (S \bigotimes S) \cdot vec(Y) \tag{5.8}$$

where $\bigotimes$ is Kronecker product, vec(Y) is a column-vector with Hadamard-transformed coefficients as input and vec(Z) represents the corresponding column vector with integer-transformed coefficients. We further define permutation matrices P_r and P_c such that

$$I_{16} = P_r \cdot (P_r)^T = (P_r)^T \cdot P_r = P_c \cdot (P_c)^T = (P_c)^T \cdot P_c \tag{5.9}$$

from Equation 5.8, we can write as follows:

$$vec(Z) = P_r^T \cdot M \cdot P_c^T \cdot vec(Y) \tag{5.10}$$

where

$$M = P_r \cdot (S \bigotimes S) \cdot P_c \tag{5.11}$$

and

$$M = \begin{pmatrix} I_4 & O_4 & O_4 & O_4 \\ O_4 & M_1 & O_4 & O_4 \\ O_4 & O_4 & M_1 & O_4 \\ O_4 & O_4 & O_4 & M_2 \end{pmatrix},$$

Similarly, I_4, O_4, M_1, and M_2 follow:

$$I_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$O_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

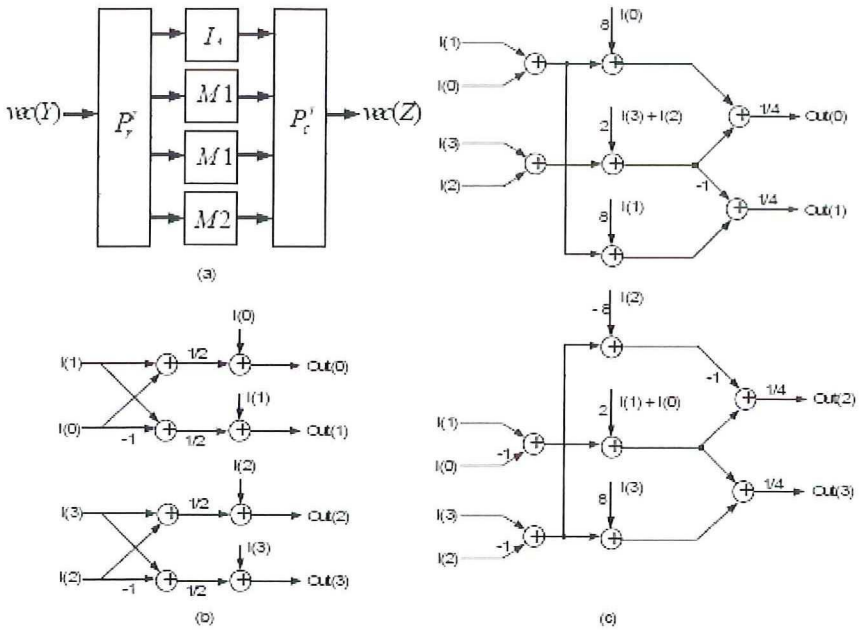


Figure 5.4: (a) Functional block diagram for 2-D transform, (b) Signal flow graph for M_1 , (c) Signal flow graph for M_2

$$M_1 = \begin{pmatrix} 1.5 & 0.5 & 0 & 0 \\ -0.5 & 1.5 & 0 & 0 \\ 0 & 0 & 1.5 & 0.5 \\ 0 & 0 & -0.5 & 1.5 \end{pmatrix},$$

$$M_2 = \begin{pmatrix} 2.25 & 0.25 & 0.75 & 0.75 \\ 0.25 & 2.25 & -0.75 & -0.75 \\ -0.75 & 0.75 & 2.25 & -0.25 \\ -0.75 & 0.75 & -0.25 & 2.25 \end{pmatrix}$$

Functional block diagram for Equation 5.11 is depicted in Figure 5.4(a). Signal flow graphs for M_1 and M_2 are depicted in Figure 5.4(b) and Figure 5.4(c) respectively. The hardware implementation for Equation 5.11 costs 30 adders only. The number of adders are reduced to a value less than half (64 vs. 30), when compared with existing state-of the art fast 2-D integer transform hardware architectures in literature. The proposed solution not only requires less area for its hardware implementation at one hand, it but consumes significantly less dynamic power on the other, because of reduced number of adder units.

5.2 Forward Integer Transform for Video Processing Applications

The intra prediction process, as explained in Section4.1, requires reconstructed pixels from neighbor blocks. Therefore, intra prediction process for next block is not started until the current block in not reconstructed and its pixels are not available for prediction. Video processing applications using intra-frames only, therefore, require efficient processing chain for real-time processing of high-resolution videos. A low-latency and high-throughput forward integer transform unit is very important requirement for an efficient intra frame processing chain. The straightforward solution for low-latency is to implement forward integer transform unit in parallel with intra-prediction mode decision unit. Depending on the outcome of intra-prediction mode decision unit, the forward integer transformed block for the selected intra-prediction mode is readily available for the next processing unit in the intra-frame processing chain with zero latency. This approach provides an efficient solution for latency issue. The area cost for this proposed solution is reduced by reusing intermediate results from Hadamard transform unit in the intra-frame processing chain. With our proposal, not only the number of operations to realize integer transform are significantly reduced (32 vs. 64) but effective latency penalty diminishes as the proposed computation for the integer transform are removed from the critical path.The remainder of this section describes our approach to identify common intermediate results and the hardware design for its implementation.

5.2.1 2-D Forward Integer Transform Algorithm

From Equations 5.2 and 5.5, the 2-D 4×4 integer transform and 2-D Hadamard transform can be expressed as follows:

$$vec(Z) = (C \bigotimes C) \cdot vec(X) \tag{5.12}$$

$$vec(Y) = (H \bigotimes H) \cdot vec(X) \tag{5.13}$$

where $\bigotimes$ is Kronecker product, vec(X) is a column-vector with residual pixel data, vec(Y), and vec(Z) are corresponding column-vectors with Hadamard-transformed coefficients and integer-transformed coefficients respectively. We

can rewrite Equation 5.12 as follows:

$$vec(Z) = vec(Y) + (vec(Z) - vec(Y)) \tag{5.14}$$

$$\therefore => vec(Z) = vec(Y) + O_{offset} \tag{5.15}$$

The above equation suggests that integer-transformed coefficients can be computed by adding an appropriate offset to Hadamard-transformed coefficients. Using Equations 5.12, 5.13 and 5.15, this appropriate O_{offset} is computed as follows:

$$O_{offset} = Q \cdot vec(X) \tag{5.16}$$

where

$$Q = (C \otimes C) - (H \otimes H) \tag{5.17}$$

Q is the difference of Kronecker product of integer and Hadamard transform matrices and are defined in Equations 5.3 and 5.5, respectively.

$$Q = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\
3 & 1 & -1 & -3 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -3 & -1 & 1 & 3 \\
1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 \\
1 & -3 & 3 & -1 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 & -1 & 3 & -3 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & -3 & -1 & 1 & 3 & 3 & 1 & -1 & -3 & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & -1 & 3 & -3 & 1 & 1 & -3 & 3 & -1 & 0 & 1 & -1 & 0
\end{pmatrix} \tag{5.18}$$

We further define permutation matrices P_{r1}, P_{r2}, and P_{r3} such that:

$$I_{16} = P_{r1} \cdot (P_{r1})^T = (P_{r1})^T \cdot P_{r1} \tag{5.19}$$

$$I_{16} = P_{r2} \cdot (P_{r2})^T = (P_{r2})^T \cdot P_{r2} \tag{5.20}$$

$$I_{16} = P_{r3} \cdot (P_{r3})^T = (P_{r3})^T \cdot P_{r3} \tag{5.21}$$

While considering the intermediate results generated in 2-D Hadamard transform processing and using permutation matrices defined above, we split Q in Equation 5.18 into 3 matrices $\bar{Q_1}$, $\bar{Q_2}$, and $\bar{Q_3}$ such that :

$$Q_1 = P_{r1} \cdot \bar{Q_1} \tag{5.22}$$

$$Q_2 = P_{r2} \cdot \bar{Q_2} \tag{5.23}$$

$$Q_3 = P_{r3} \cdot \bar{Q_3} \tag{5.24}$$

From Equations 5.16 and 5.17, we can write:

$$O_{offset} = (P_{r1}^T \cdot Q_1 + P_{r2}^T \cdot Q_2 + P_{r3}^T \cdot Q_3) \tag{5.25}$$

$$Q_1 = \begin{pmatrix} 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \\ 2 & 0 & 0 & -2 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -2 & 0 & 0 & 2 \\ 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 & -2 & 0 & 0 & 2 & 2 & 0 & 0 & -2 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.26}$$

$$Q_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 2 & -2 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & -2 & 2 & 0 \\ 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & -2 & 2 & 0 & 0 & 2 & -2 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.27}$$

$$Q_3 = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 \\
1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 \\
1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0
\end{pmatrix} \tag{5.28}$$

and

$$P_{r1} = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} \tag{5.29}$$

$$P_{r2} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix} \tag{5.30}$$

$$P_{r3} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} \tag{5.31}$$

From Equations 5.26 and 5.27, we can observe that Q_1 and Q_2 compute the difference (or difference scaled by a factor of 2) between 1st and 4th rows, or 2nd and 3rd rows of input block X , respectively. These row-differences are also computed in the 2-D Hadamard transform computations. Similarly, in Equation 5.28, Q_3 essentially computes the difference between rows of 1-D Hadamard transformed results. Again, these difference values are also computed in 2-D Hadamard transform computation. We propose to reuse these intermediate results from 2-D Hadamard transform computation unit and, therefore, replace input vec(X) by input vec(D) in Equation 5.16. Consequently, from Equation 5.25we can derive O_{offset} as follows:

$$O_{offset} = (P_{r1} \cdot W_1 + P_{r2} \cdot W_2 + P_{r3} \cdot W_3) \tag{5.32}$$

where W_1, W_2, and W_3 are given as follows:

$$W_1 = \begin{bmatrix} C & O_4 & O_4 & O_4 \end{bmatrix}^T \tag{5.33}$$

$$W_2 = \begin{bmatrix} O_4 & C & O_4 & O_4 \end{bmatrix}^T \tag{5.34}$$

$$W_3 = \begin{bmatrix} O_4 & O_4 & I_4 & I_4 \end{bmatrix}^T \tag{5.35}$$

and

$$vec(D) = \begin{bmatrix} D_0 & D_1 & \dots & D_{15} \end{bmatrix}^T \tag{5.36}$$

Where

$$D_0 = x_{00} - x_{30} \tag{5.37}$$

$$D_1 = x_{01} - x_{31} \tag{5.38}$$

$$D_2 = x_{02} - x_{32} \tag{5.39}$$

$$D_3 = x_{03} - x_{33} \tag{5.40}$$

$$D_4 = x_{10} - x_{20} \tag{5.41}$$

$$D_5 = x_{11} - x_{21} \tag{5.42}$$

$$D_6 = x_{12} - x_{22} \tag{5.43}$$

$$D_7 = x_{13} - x_{23} \tag{5.44}$$

$$D_8 = X_{00} - X_{03} \tag{5.45}$$

$$D_9 = X_{10} - X_{13} \tag{5.46}$$

$$D_{10} = X_{20} - X_{23} \tag{5.47}$$

$$D_{11} = X_{30} - X_{33} \tag{5.48}$$

$$D_{12} = X_{01} - X_{02} \tag{5.49}$$

$$D_{13} = X_{11} - X_{12} \tag{5.50}$$

$$D_{14} = X_{21} - X_{22} \tag{5.51}$$

$$D_{15} = X_{31} - X_{32} \tag{5.52}$$

In above set of equations, x_{ij} is a data-item from 4x4 residual pixels-block, and X_{ij} represents 1-D Hadamard-transformed coefficients for same 4×4 residual pixels-block. While i, j = 0, 1, 2 and 3; denotes row and column in 4×4 input block respectively. The signal-flow diagram for Equation 5.32 is depicted in Figure 5.5 Consequently, the computation of O_{offset} requires 20 adders for its hardware implementation. Another 12 adders are required to derive integer-transformed coefficients by adding these computed-offset values to Hadamard-transformed coefficients. t The number of adders for our proposed design to realize integer transform in intra-frame encoder are, therefore, reduced to half (64 vs. 30), when compared with existing state-of the art fast 2-D integer transform hardware architectures in literature. Moreover, the computation unit for O_{offset} operates in parallel with the Hadamard transform unit and, therefore, does not contribute to the overall latency of the intra-frame processing chain.

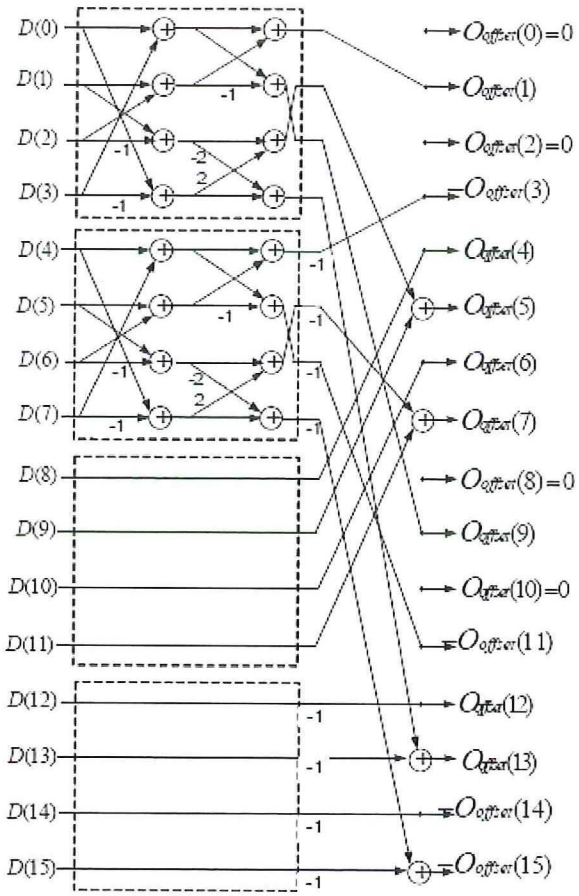


Figure 5.5: Data flow for computation of O_{offset}

5.3 Summary

In this chapter, we have presented two solutions for realization of integer transform in the processing chain of intra-frame encoder applications. The first solution targets image compression applications running on battery-powered electronic devices, such as digital still camera. The proposed solution utilizes a novel 2-D transform to derive integer-transformed coefficients directly from that of Hadamard-transformed coefficients with significantly reduced number of operations and, therefore, area and power consumption. The second solution targets real-time video compression applications using intra-frames only. The proposed solution reduces the effective latency penalty of forward integer transform to zero. Moreover, it aggressively reuses the intermediate results from Hadamard transform and, therefore, requires significantly reduced number of operations and, thus, less area for its hardware implementation.

Note.

The content of this chapter is based on the the following paper:

Muhammad Nadeem, Stephan Wong, and Georgi Kuzmanov, *An Efficient Realization of Forward Integer Transform in H.264/AVC Intra-frame Encoder* , proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS) 2010, pp. 71-78, Samos, Greece, July 2010.

6

Inverse Integer Transform

THE H.264/AVC is a block-based, hybrid video coding standard that utilizes a spatial, DCT-based integer inverse-transform design in its compression engine to avoid encoder-decoder mismatch. Although, the transformation can be carried out by addition operations only, it is one of the top 3 compute-intensive processing modules in H.264/AVC decoder. In this chapter, we propose two hardware designs for inverse integer transform unit in H.264/AVC compliant video encoder and decoder.

The main contributions of this chapter are:

1. A proposal to compute inverse integer transform on-the-fly and a low-latency hardware design for inverse integer transform unit in an intra-frame encoder processing chain.

2. Derivation of data-driven algorithm for inverse integer transform in H.264/AVC with variable number of operations.

3. Low-power, high-throughput hardware design for inverse integer transform unit based on data-driven algorithm.

The chapter is organized as follows: In Section, 6.1, we briefly provide a motivation for low-latency inverse integer transform. Subsequently, a hardware design for low-latency inverse integer transform is provided in the same section. In Section 6.2, a data-driven algorithm for inverse integer transform is described. A low-power, high-throughput hardware design based on data-driven algorithm is also presented at the end of this section. Finally, Section 6.3 summarizes the chapter.

6.1 Low-Latency Inverse Integer Transform

In this section, we first describe our approach for the low-latency and area-efficient solution for the inverse integer transform unit. Subsequently, the algorithm and the corresponding architecture for the inverse integer transform unit are described.

A typical implementation of forward and inverse quantization units, from the perspective to reduce multiplier's area-cost, processes the input data either serially or four data-items in parallel. Similarly, the hardware unit for inverse integer transform, depending upon implemented algorithm, requires one-fourth, half, or a complete 4×4 input block of data to start its processing. Therefore, an additional buffer is required for pipeline its implementation or a delay is to be introduced before start of inverse integer transform process. Moreover, most of inverse integer transform algorithms assume that all 4×4 input block data-items are non-zero. Consequently, these algorithms perform constant number of operations per block in order to compute inverse integer transform for a given block of input data.

However, in digital video coding, it is very common that a substantial number of spatial high-frequency transformed-coefficients of residual block are quantized to zero. A typical transformed and quantized residual block contains few non-zero coefficients in the low-frequency region and mostly zero-valued coefficients in the high-frequency region. From inverse integer transform unit's perspective, while decoding incoming video bitstream in decoder or during (inverse) quantization process in encoder, it is already known for that how many and at what positions in a 4×4 input data-block, the transformed-quantized coefficients are non-zero and, therefore, how shall they contribute to the pixel values for the reconstructed block. A considerable amount of computations can be saved by exploiting this information, while developing an algorithm for inverse integer transform.

In this work, we have followed a different approach and derived a serial algorithm for inverse integer transform unit. The hardware unit for inverse integer transform, based on serial processing algorithm, directly interfaces with the inverse quantization unit, without requiring any additional buffer or delay. Inverse integer transform unit processes the incoming inverse-quantized data items on-the-fly and accumulates their contribution towards the final value for the inverse integer transformed block or residual block. With this approach, latency penalty is significantly reduced. In case of zero-valued incoming data-item, the processing for this particular data-item is skipped in order to avoid

unnecessary signal activity in the circuit and to potentially reduce the dynamic power consumption.

6.1.1 Low-latency Inverse Integer Transform Algorithm

In the H.264/AVC video coding standard, for any 4×4 inverse quantized input block X, the corresponding residual pixel block Y is computed as follows:

$$Y = C_i \cdot X \cdot C_i^T \tag{6.1}$$

where

$$C_i = \begin{pmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{pmatrix} \tag{6.2}$$

Direct 2-D inverse integer transform for 1-D inverse integer transform in Equation.6.1 can be computed as follows:

$$vec(Y) = (C_i \bigotimes C_i) \cdot vec(X) \tag{6.3}$$

In Equation 6.3, $\bigotimes$ is Kronecker product, vec(X) is a column-vector consists of inverse-quantized data-items as input and vec(Y) represents the corresponding column-vector with inverse integer transformed coefficients as output. From Equations. 6.1 and 6.3, we can write:

$C_i \bigotimes C_i =$

$$\begin{pmatrix}
1 & 1 & 1 & 1/2 & 1 & 1 & 1 & 1/2 & 1 & 1 & 1 & 1/2 & 1/2 & 1/2 & 1/2 & 1/4 \\
1 & 1/2 & -1 & -1 & 1 & 1/2 & -1 & -1 & 1 & 1/2 & -1 & -1 & 1/2 & 1/4 & -1/2 & -1/2 \\
1 & -1/2 & -1 & 1 & 1 & -1/2 & -1 & 1 & 1 & -1/2 & -1 & 1 & 1/2 & -1/4 & -1/2 & 1/2 \\
1 & -1 & 1 & -1/2 & 1 & -1 & 1 & -1/2 & 1 & -1 & 1 & -1/2 & 1/2 & -1/2 & 1/2 & -1/4 \\
1 & 1 & 1 & 1/2 & 1/2 & 1/2 & 1/2 & 1/4 & -1 & -1 & -1 & 1/2 & -1 & -1 & -1 & -1/2 \\
1 & 1/2 & -1 & -1 & 1/2 & 1/4 & -1/2 & -1/2 & -1 & -1/2 & 1 & 1 & -1 & -1/2 & 1 & 1 \\
1 & -1/2 & -1 & 1 & 1/2 & -1/4 & -1/2 & 1/2 & -1 & 1/2 & 1 & -1 & -1 & 1/2 & 1 & -1 \\
1 & -1 & 1 & -1/2 & 1/2 & -1/2 & 1/2 & -1/4 & -1 & 1 & -1 & 1/2 & -1 & 1 & -1 & 1/2 \\
1 & 1 & 1 & 1/2 & -1/2 & -1/2 & -1/2 & -1/4 & -1 & -1 & -1 & -1/2 & 1 & 1 & 1 & 1/2 \\
1 & 1/2 & -1 & -1 & -1/2 & -1/4 & 1/2 & 1/2 & -1 & -1/2 & 1 & 1 & 1 & 1/2 & -1 & -1 \\
1 & -1/2 & -1 & 1 & -1/2 & 1/4 & 1/2 & -1/2 & -1 & 1/2 & 1 & -1 & 1 & -1/2 & -1 & 1 \\
1 & -1 & 1 & -1/2 & -1/2 & 1/2 & -1/2 & 1/4 & -1 & 1 & -1 & 1/2 & 1 & -1 & 1 & -1/2 \\
1 & 1 & 1 & 1/2 & -1 & -1 & -1 & -1/2 & 1 & 1 & 1 & 1/2 & -1/2 & -1/2 & -1/2 & -1/4 \\
1 & 1/2 & -1 & -1 & -1 & -1/2 & 1 & 1 & 1 & 1/2 & -1 & -1 & -1/2 & -1/4 & 1/2 & 1/2 \\
1 & -1/2 & -1 & 1 & -1 & 1/2 & 1 & -1 & 1 & -1/2 & -1 & 1 & -1/2 & 1/4 & 1/2 & -1/2 \\
1 & -1 & 1 & -1/2 & -1 & 1 & -1 & 1/2 & 1 & -1 & 1 & -1/2 & -1/2 & 1/2 & -1/2 & 1/4
\end{pmatrix} \tag{6.4}$$

Equation 6.4 suggests that the residual pixels-block Y can be computed by simply adding (or subtracting) current value (or scaled by $1/2$, or $1/4$) of each

given input coefficient X_{ij} to the accumulated values for the output block data-items Y_{ij}. Therefore, the serial inverse integer transform algorithm requires only one register to hold incoming input data-item and 16 registers to hold partial or accumulated results for the output block data-items Y_{ij}. Similarly, 16 add/sub units are required to add the current (or scaled) value to the already accumulated partial results. The accumulator is initialized to zero at the start of each input block. As soon as the last data-item of the input block arrives, the corresponding residual pixel block Y is available at the output of the inverse integer transform unit. With this approach, inverse-quantized data-items X_{ij} from the inverse quantization unit are processed on-the-fly without introducing any delay for the computation of the residual pixel block. Moreover, only non-zero input data-items are stored in the input register of inverse integer transform unit for processing. This helps to reduce the unnecessary signal activity in the circuit which can, therefore, contribute to potentially reduce the dynamic power consumption especially at coarser quantization parameter (Qp) value and/or smoother regions of the video frame being processed.

6.1.2 Low-latency Inverse Integer Transform Hardware Design

In this section, high-level organization of low-latency inverse integer transform unit is introduced. The design is based on inverse integer transform algorithm introduced in Section 6.1.1. Subsequently, hardware design of basic processing unit along with control unit is explained.

High Level Organization The functional block diagram of inverse integer transform unit is depicted in Figure 6.1. The proposed design takes inverse-quantized coefficient (X_val) and its position in the input-block as inputs to the hardware unit. In case this input inverse-quantized coefficient is non-zero, it is stored in a register (InReg) as depicted in Figure 6.1. P_x processing blocks hold the partial or final computed value of each residual-pixel-bock data-items Y_{ij}. There are 16 such processing blocks in order to compute a complete 4×4 residual pixels-block.

Hardware Design for Basic Processing Unit The internal design of $P_x$block is depicted in Figure 6.2. This block consists of one multiplexer (Mux) at the input, one accumulator register Acc, and one Add/Sub unit. Mux is used to select the correct contribution of input inverse-quantized data-item to compute Y_{ij}, while Acc register holds partial results during processing. Reset control

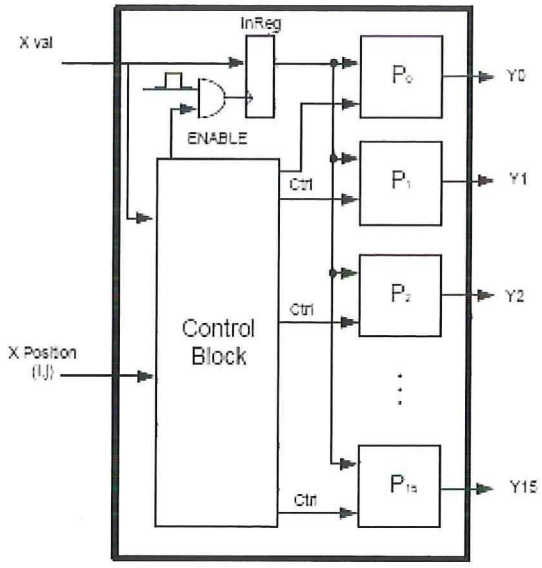


Figure 6.1: Functional block diagram for low-latency inverse integer transform unit in H.264/AVC.

signal initializes Acc to zero at the start of each 4×4 input block during computation of inverse integer transform coefficients..

Control Unit The control block in Figure 6.1 generates signals in order to control data-flow in the processing unit. Control signals for Mux and Add/Sub can be straightforwardly derived from Equation 6.4. Enable control signal is used to provide gated-clock to InReg. The generation of Enable and Reset control signals is depicted in Figure 6.3 .

6.2 Low-power, High-throughput Inverse Integer Transform

In this section, we first propose to categorize 4×4 input data-block into a set of 4 different types along with its justification. Subsequently, based on the proposed input data-block types, a data-driven algorithm for inverse integer transform is described in detail.

As mentioned earlier, a typical transformed block after quantization process

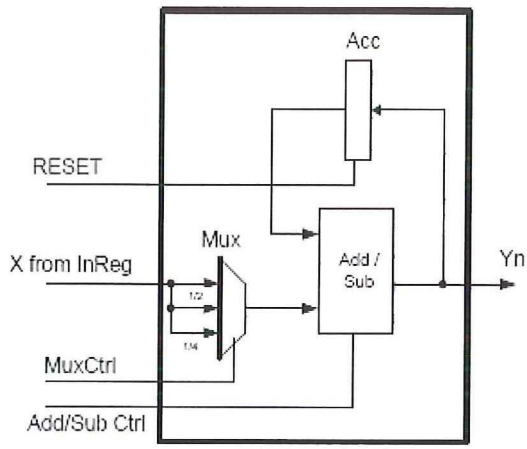


Figure 6.2: Basic processing unit design for low-latency inverse integer transform.

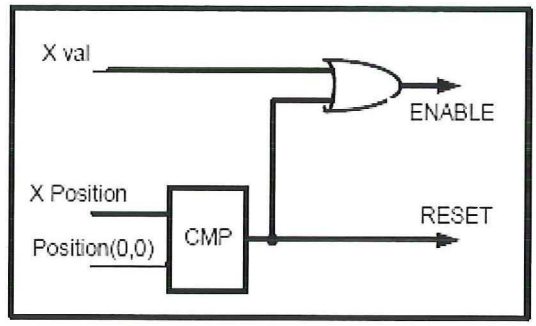


Figure 6.3: Control unit design for low-latency inverse integer transform.

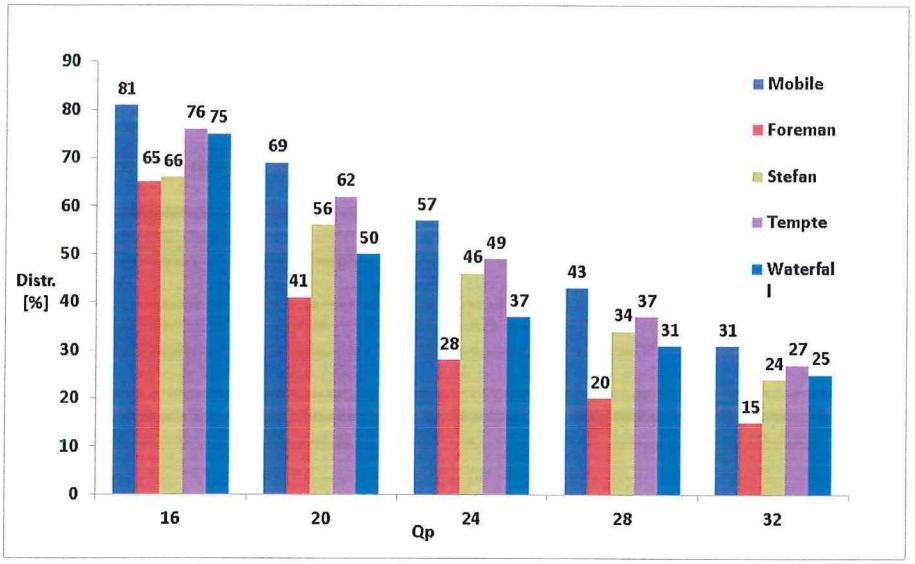
consists few non-zero coefficients. The distribution of these non-zero coefficients , beside various factors like video signal itself and encoding type, etc., heavily depends on the Quantization parameter (Qp) used for encoding of this block. Though ta coarser quantization parameter provides better compression ratio because of few non-zero blocks after quantization process, the quality of compressed video suffers in terms of PSNR value. This is the reason that the primary responsibility of the rate control in an encoder is to allocate bits in a video frame such that the generated compressed video bitstream meets available bandwidth constraint while providing the best possible video quality. Consequently, as Qp varies from a coarser value to a fine value, number of non-zero blocks after the quantization process increases. This is also depicted in Figure 6.4a for a number of video test sequences (with 352×288, CIF resolution) encoded using typical typical Qp values range.

A typical distribution of non-zero transformed-quantized block is such that it contains few non-zero coefficients in the low-frequency region and mostly zero-valued coefficients in the high-frequency region. In our work, we propose to categorize these input transformed-quantized blocks into a set of 4 different types, namely All-Zero blocks (AZB), DC-only blocks, Upper Left Triangular (ULT) blocks and Normal blocks. An example of such blocks is depicted in Figure 6.6. The AZB type can be easily justified from the non-zero block distribution in Figure 6.4a as for test video sequences, profile data suggests that the percentage of AZB blocks varies between 30% to 75% for a typical range of Qp values (16 to 32). Similarly, percentage distribution of DC-only blocks, upper left Triangular blocks and normal blocks, within non-zero blocks is depicted in Figures 6.4b, 6.5a, and 6.5b respectively, for the same video test sequences encoded using same set of Qp values.

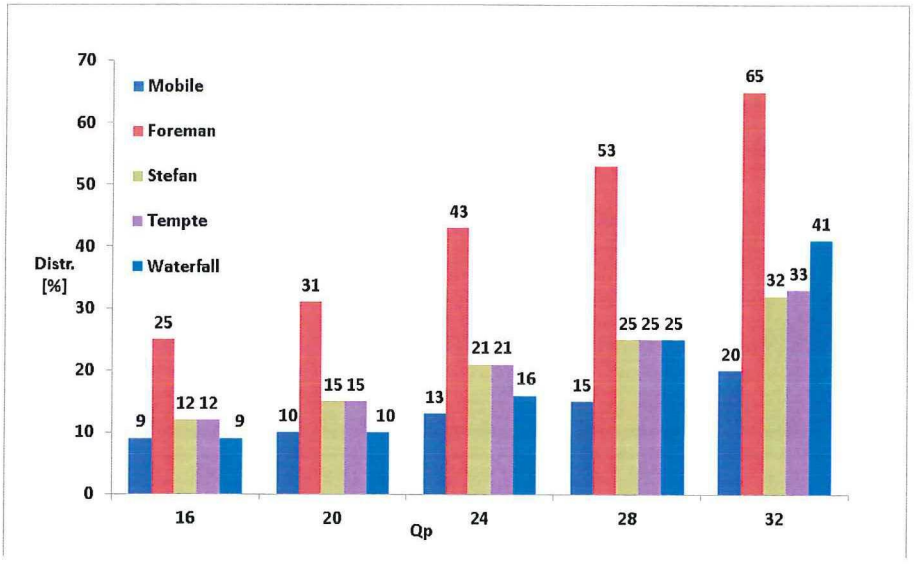
6.2.1 Data-driven Algorithm For Inverse Integer Transform

Signal-flow diagram for 1-D inverse integer transform, with four input data-items, is depicted in Figure 6.7(d). For scenarios with only one, two, or three non-zero data-items in the input data vector, the signal-flow diagrams for 1-D inverse integer transform can be derived by removing the corresponding signal-flow part in Figure 6.7(d). Signal-flow diagrams for 1-D inverse integer transform with only one, two, or three non-zero coefficients is depicted in Figures 6.7(a), 6.7(b), and 6.7(c), respectively. The resulted signal-flow diagrams suggest that the number of addition operations for 1- D inverse integer transform is reduced by 100%, 50%, and 25%, respectively.

We further denote signal-flow for 1-D inverse integer transform with one, two,

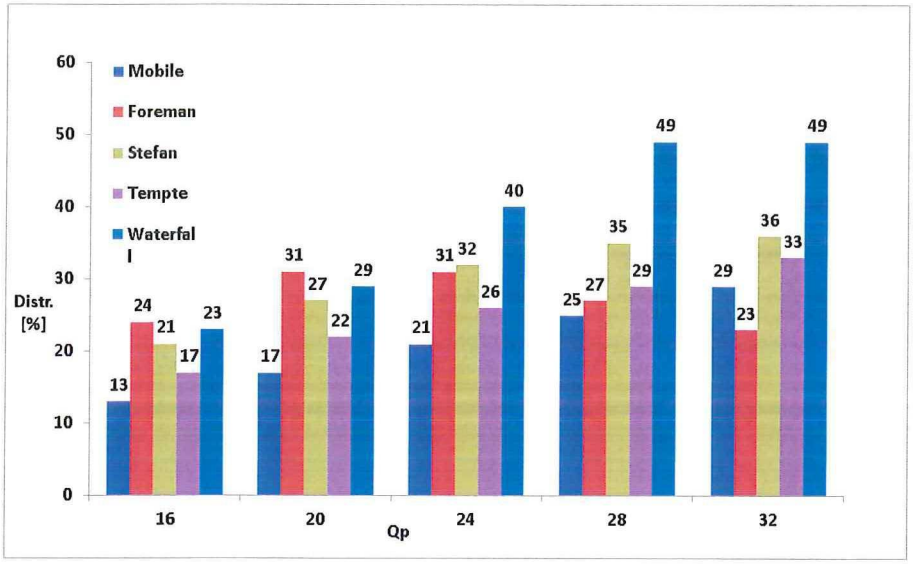


(a)

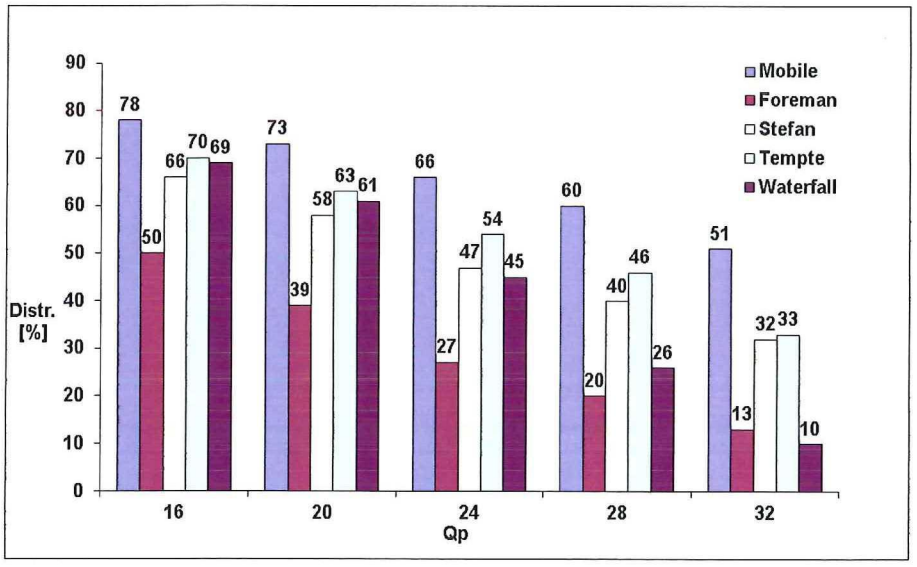


(b)

Figure 6.4: Percentage distribution of 4×4input block types for inverse integer transform: (a) Non-zero blocks , (b) DC-blocks.



(a)



(b)

Figure 6.5: Percentage distribution of 4×4input block types for inverse integer transform: (a) ULT-blocks, (b) Normal-blocks.

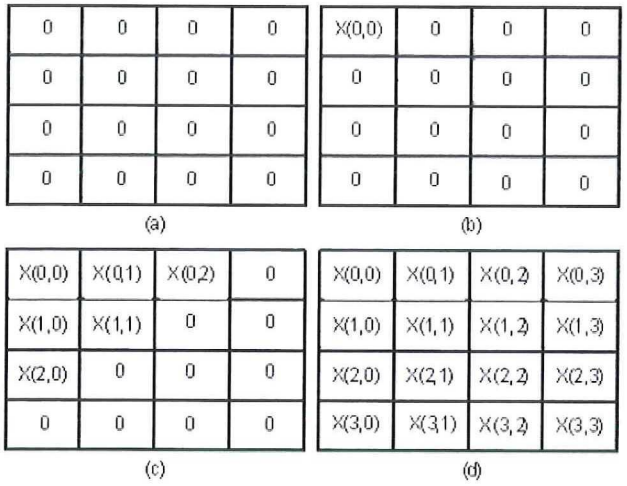


Figure 6.6: Proposed input block types: (a) All-zero block, (b) DC-block, (c) ULT-block, (d) Normal block.

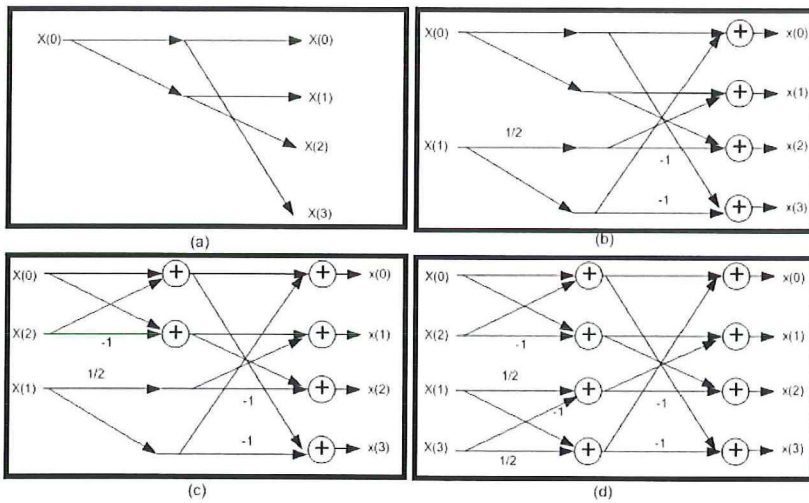


Figure 6.7: Signal-flow diagrams for 1-D inverse integer transform cases: (a) Case M1: single non-zero input data-item , (b) Case M2: two non-zero input data-items, (c) Case M3: three non-zero input data-items and, (d) Case M4: four non-zero input data-items.

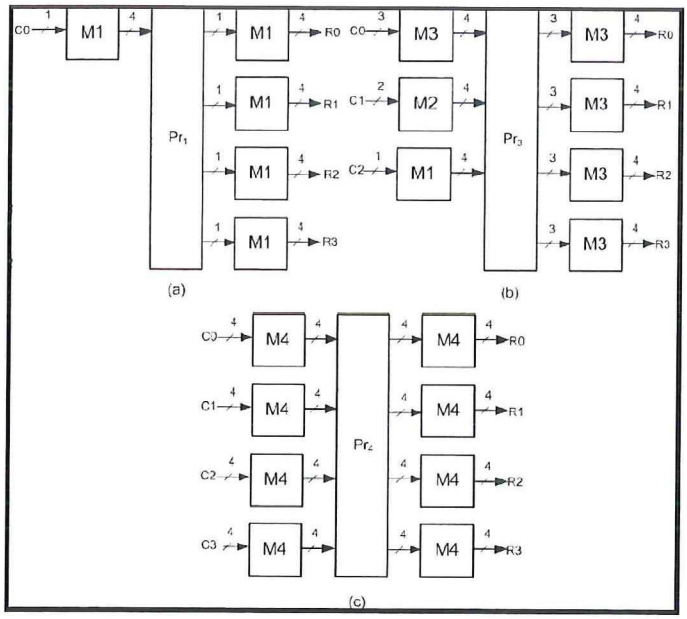
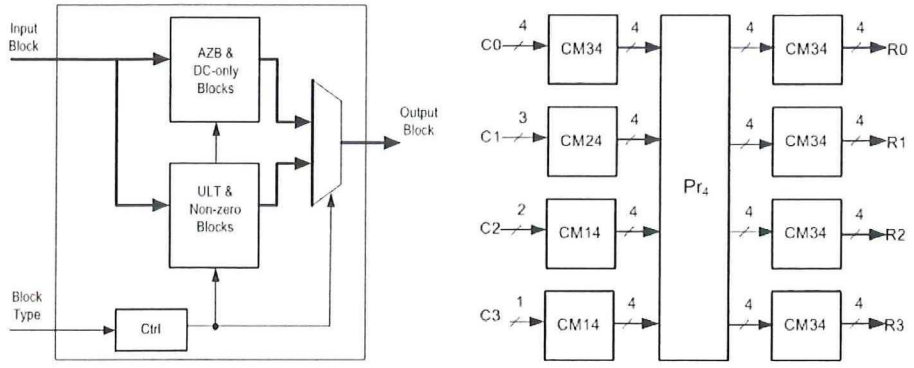


Figure 6.8: Inverse integer transform unit functional block diagram for: (a) DC, (b) ULT and, (c) Normal blocks.

three, or four non-zero data-items in Figure6.7 as processing units M1, M2, M3, and M4 respectively. Based on these processing units, the functional block diagram for the 2-D inverse integer transform for the DC-only, ULT, and the normal block is depicted in Figures 6.8(a), 6.8(b), and 6.8(c), respectively. Where, C_x in Figure 6.8, is input column-vector and R_x denotes corresponding 2-D inverse-transformed row-vector for a given input block. P_{rx} represents row-column permutations to realize transpose between two 1-D inverse integer transform units. 2-D inverse integer transform for a normal 4×4 input block requires 64 addition operations (Figure 6.7(d) and Figure 6.8(c)). This number is reduced to 34 and 0 for AZB and DC-only input block types respectively.

A straightforward hardware design for inverse integer transform is to implement independent units for the identified block types (i.e. AZB, DC-only, ULT and normal input blocks). This approach potentially reduces the dynamic power consumption, at the cost of additional area/resources for 34 adders in ULT processing unit and control circuitry to activate appropriate processing units accordingly. We propose to reduce this area-overhead by designing a configurable hardware design for inverse integer transform and, therefore, sharing hardware resources between identified independent processing units.



(a) (b)

Figure 6.9: Inverse integer transform unit (a) High-level organization, (b) Functional block diagram.

6.2.2 Configurable, Low-power Inverse Integer Transform Unit Hardware Design

In this section, we present a hardware design for the data-driven inverse integer transform unit using configurable processing block.

High Level Organization High-level organization for data-driven inverse integer transform unit is depicted in Figure 6.9a. If the input block is of category AZB or a DC-only, the control unit routes it to the top level processing block (Figure 6.9a) which is essentially a bypass stage as no further processing is required. The value for all the data-items in the output block is either zero or DC value for AZB or DC-only block types, respectively.

The proposed design implements computation for AZB consists of takes 4×4 input block X, along with input-block type information determined during the entropy decoding process. If the input block is an AZB or a DC-only block, the control unit routes it to the top level processing block (Figure 6.9a) which is essentially a bypass stage as no further processing is required for such blocks. In such cases, the value for all the data items in the output block is either zero or the DC value for the AZB or the DC-only block types, respectively.

ULT and normal (non-zero) input blocks are routed to the lower processing block in Figure 6.9a. The internal organization of this block is depicted in Figure 6.8. Since the processing blocks M1-M3 are derived from M4 (Figure 6.7)

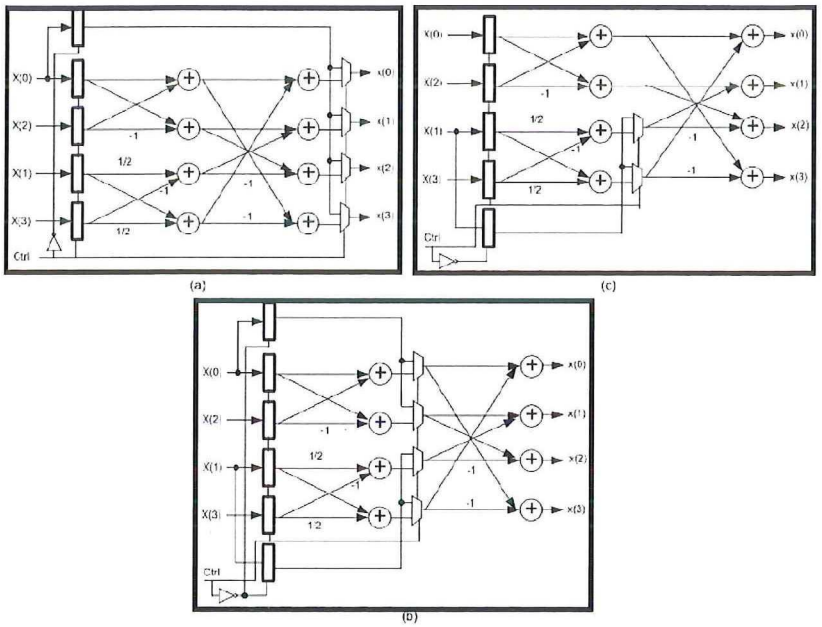


Figure 6.10: Data-flow diagrams for configurable 1-D inverse integer transform units: (a) CM14, (b) CM24 and, (c) CM34.

and have similar structure, therefore, we can design a configurable processing units (CM14, CM24, and CM34) with overlapped datapath to reduce the hardware resources for its implementation. The configurable processing units (CM14, CM24, and CM34) as the name suggest can be configured to provide processing for either (M1, M4), (M2, M4), or (M3, M4). The internal design for these configurable units is depicted in Figures 6.10(a) - 6.10(c). With these configurable processing units, the overhead of 34 adders for ULT is removed completely.

6.3 Summary

In this chapter, we have presented two hardware designs for inverse integer transform in H.264/AVC video codec. The first design is intended for intra-frame encoder with significantly reduced latency. The proposed design processes input data on-the-fly in order to compute inverse integer transformed data block.

A data-driven algorithm for inverse integer transform, with variable number

of operations, is also introduced in this chapter. The proposed hardware design,based on data-driven algorithm, provides high-throughput and consumes significantly less dynamic power for its implementation. The area-overhead of inverse integer transform unit is reduced by designing a configurable hardware and efficiently sharing hardware resources between independent processing units.

Note.

The content of this chapter is based on the the following papers:

Muhammad Nadeem, Stephan Wong, and Georgi Kuzmanov, *Inverse Integer Transform in H.264/AVC Intra-frame Encoder*, proceedings of International Symposium on Electronic Design, Test and Application 2011 (DELTA), pp. 228-233, Queens town, New Zealand, January 2011.

Muhammad Nadeem, Stephan Wong, and Georgi Kuzmanov, *Configurable, Low-power design for Inverse Integer Transform in H.264/AVC*, proceedings of International Conference on Frontiers of Information Technology 2010 (FIT), pp. 32-37, Islamabad, Pakistan, December 2010.

7

Custom Operations

APPLICATION-specific embedded system design has become more difficult than ever due to rapid increase in design complexity. Efficiency and flexibility must be carefully balanced to meet a number of embedded system application requirements. Applications running on a programmable platform can be executed either as a software algorithm or a specialized hardware unit. The first approach is slowest one but most flexible. Specialized hardware approach, on the other hand, is the fastest approach but least flexible. From video processing perspective, flexibility to adapt to video coding standard evolutions and market/technology induced changes has become a new dimension in the algorithm/architecture design. Newly emerged reconfigurable and extensible processors offer a favorable tradeoff between efficiency and flexibility, and a promising way to minimize certain important metrics (e.g., execution-time and code-size, etc.) of embedded processors. Reconfigurable computing has proven itself to be able to speed-up many applications despite its lack in achieving high frequencies. However, frequency is not the sole factor that determines performance. Field Programmable Gate Arrays (FPGAs) - as most utilized reconfigurable fabric nowadays - provide a large amount of parallel structures those when exploited efficiently greatly contribute to the speedup of applications. Therefore, reconfigurable computing could possibly be the solution to provide needed flexibility along with performance.

With high-performance computing point of view, a custom instruction in a processor is an assembly operation that implements functionality of a compute-intensive kernel of an application for its accelerated performance. Traditionally, custom operations have already been used in commercial Complex Instruction Set Computers (CISC) and Digital Signal Processors (DSP) to accommodate operations that consist of more complexity; operations such as floating point or complex arithmetic operations. By extending instruction set with a more complex operation, fewer operations are needed to achieve the

same result leading to a more compact set of operations in an application with potentially faster execution time as a result. From efficient and flexible video processing perspective, we propose to incorporate a customized functional unit in the datapath of reconfigurable, soft-core, VLIW processor for compute intensive functions in H.264/AVC video codec. More specifically, main contributions are provided as follows:

1. A proposal to extend ISA of ρ-VEX reconfigurable, soft-core, VLIW processor with an application-specific custom-operations.
2. Designs for various compute-intensive operations in H.264/AVC video codec and their implementation in ρ-VEX datapath.

The chapter is organized as follows: In Section 7.1, we provide an overview of proposed customized functional unit in ρ-VEX, soft-core VLIW processor. A number of custom-operations for compute-intensive functions in H.264/AVC are presented in Section 7.2. Finally, Section 7.3 provides summary of this chapter.

7.1 ρ-VEX: Customized VLIW Soft-core Processor

Digital video signal processing applications are identified to have significant fine and coarse-grained parallelism [13] Therefore, Very Long Instruction Word (VLIW) processors are preferred computing platform for such applications to achieve high performance [14]. VLIW processors exploit existing parallelism in video processing applications in order to provide improved performance by utilizing pipelining and multiple functional units (FUs) to execute several operations simultaneously. For a moderate cost, the performance of VLIW processors can be further improved. ρ-VEX is an open source, extensible, and reconfigurable, soft-core VLIW processor. We propose to incorporate a customized functional unit in the datapath of ρ-VEX VLIW processor. This customized functional unit implements application-specific custom instruction for compute-intensive functions in video codec H.264/AVC.

The top-level functional blocks in the five-stage pipeline of ρ-VEX along with Custom Functional Unit (CFU) is depicted in Figure 7.1. The ISA of VEX processor is extendable. CFU implements new application-specific custom-operations with a latency of 2 cycles. The CFU is part of Execution stages (Exec.1 and Exec. 2) in the pipeline of ρ-VEX processor and interfaces with Decode and WriteBack stages as depicted in Figure 7.1. The custom functional

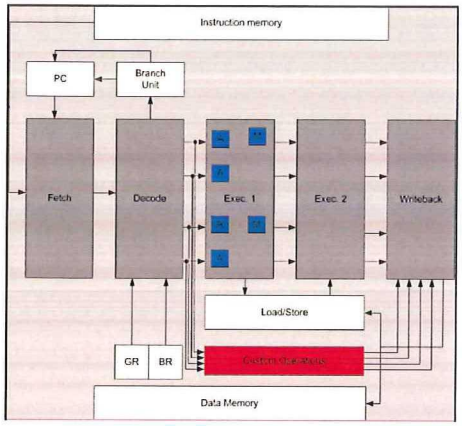


Figure 7.1: ρ-VEX Pipeline Organization with Customized Functional Unit

unit takes operands from General-Purpose Register File (GR) and produces results in the destination register in GR. The op-code decoded in Decode stage, along with the source-register identifiers are passed on to CFU. Similarly, the computed results, along with destination-register identifiers are delivered to WriteBack stage after execution.

7.2 Custom Operations for H.264/AVC Video Codec

In this section, we presents couple of custom operations for compute-intensive functions in H.264/AVC video codec. More specifically, we propose custom operations for deblocking filter, intra prediction, forward integer transform, inverse integer transform and Hadamard transform in H.264/AVC. These custom operations are implemented in CFU and datapath for these custom operations is designed using optimized algorithms with significantly reduced number of operations as described in previous chapters of this dissertation. In the remainder of this section, the design of proposed custom operations is explained in detail.

7.2.1 Custom Operation for Deblocking Filter

As mentioned in Chapter 3, the deblocking filter operation is identical along horizontal and vertical edges in a macroblock. Therefore, a single custom operation is proposed to compute filtered-pixels across both vertical as well as a horizontal edge, one at a time. The datapath for this custom operation is

implemented using optimized algorithm as listed in Algorithm 3.2. The syntax for deblocking filter custom operation is provided below:

Syntax:

$$CU\_DBF\_H264 \quad rsrc_1 \, , rsrc_2 \, , rsrc_3 \rightarrow rdst_1 \, , rdst_2 \tag{7.1}$$

Function:

Inputs :
$rsrc_1$: *pixels* $p0 - p3$
$rsrc_2$: *pixels* $q0 - q3$
$rsrc_3$: *Filter controls* (*Alpha*, *Beta*, *Bs*, *Tc0*):

Outputs :
$rdst_1$: *filtered pixels* $p0 - p3$
$rdst_2$: *filtered pixels* $q0 - q3$

Attributes:

The proposed custom operation for deblocking filter is implemented using pipeline fashion and occupy two issue-slots. The other important attributes, such as opcode and latency etc., is provided in Table 7.1.

Table 7.1: Custom Operation: Deblocking Filter

Functional Unit	Custom Unit H264
Operation Code	110
Number of Operands	3
Latency	2
Issue Slots	2
Modifier	None

7.2.2 Custom Operation for Intra-prediction

There are 9 intra-prediction modes for 4×4 luma block-type in H.264/AVC with varying complexity. Computation for these intra-prediction modes requires up to 13 pixels from left, top and top-right neighbor 4×4 blocks as depicted in Figure 4.2. The intra-predicted pixels-block consists of 16, 8-bits pixels. For 32-bit registers in GR, the computed intra-predicted pixels-block

can not be transferred out to GR using single custom operation. We, therefore, propose two custom operations for intra-prediction in H.264/AVC. The first custom operation computes top two pixels-rows, while, the second custom operation returns reaming bottom two pixels-rows in a 4×4 intra-predicted pixels-block. The proposed custom operations do not support intra-prediction modes for 16×16 luma and 8×8 chroma block types. The datapath is implemented using decomposed filter kernel equations as listed in Algorithm 4.1. The syntax for intra-prediction custom operations is provided below:

Syntax:

$$CU\_INTRA\_PRED\_HI\_H264 \quad rsrc_1, rsrc_2, rsrc_3, rsrc_4 \rightarrow rdst1, rdst2 \tag{7.2}$$

$$CU\_INTRA\_PRED\_LO\_H264 \quad rsrc_1, rsrc_2, rsrc_3, rsrc_4 \rightarrow rdst1, rdst2 \tag{7.3}$$

Function:

Inputs :
$rsrc_1$: 4 − *neighbour pixels from top* 4×4 *block*
$rsrc_2$: 4 − *neighbour pixels from top right* 4×4 *block*
$rsrc_3$: 1 − *neighbour pixels from top left* 4×4 *block*
$rsrc_4$: 4 − *neighbour pixels from left* 4×4 *block*

Outputs : (CU_INTRA_PRED_HI_H264)
$rdst_1$: *predicted pixels for first row*
$rdst_2$: *predicted pixels for second row*

Outputs : (CU_INTRA_PRED_LO_H264)
$rdst_1$: *predicted pixels for third row*
$rdst_2$: *predicted pixels for fourth row*

Attributes:

We proposed two custom operations for intra-prediction module. The datapath for both custom operations is same except the final stage of predicted-pixels selection. The first custom operation, with opcode 111, returns top two pixels-rows, while, the second custom operation with opcode 112 returns reaming bottom two pixels-rows in a 4×4 intra-predicted pixels-block. The proposed

custom operation for intra-prediction is implemented using pipeline fashion and occupy two issue-slots. The other important attributes, such as opcode and latency etc., is provided in Table 7.2.

Attributes:

Table 7.2: Custom Operation: Intra-Prediction

Functional Unit	Custom Unit H264
Operation Code	111,112
Number of Operands	4
Latency	2
Issue Slots	2
Modifier	None

7.2.3 Custom Operation for forward/inverse Integer Transform and Hadamard Transform

H.264/AVC video coding standard utilizes multiple, separable, spatial transforms in its video processing chain. . The transformations are applied on 4×4 blocks of data. Forward and inverse integer transforms are defined in Equation 5.3 and Equation 6.2 respectively, while Hadamard transform is provided in Equation 5.5. We propose two custom operations to compute forward and inverse integer transforms, while one custom operation for 4×4 Hadamard transform. 4.1. The syntax for intra-prediction custom operations is provided below:

Syntax:

$$\textit{CU\_FW\_INTG\_TRANS\_H264} \quad rsrc1, rsrc2 \rightarrow rdst1, rdst2 \tag{7.4}$$

$$\textit{CU\_INV\_INTG\_TRANS\_H264} \quad rsrc1, rsrc2 \rightarrow rdst1, rdst2 \tag{7.5}$$

$$\textit{CU\_HADAMARD\_TRANS\_H264} \quad rsrc1, rsrc2 \rightarrow rdst1, rdst2 \tag{7.6}$$

Function:

Inputs :
$rsrc_1$: $[x_0\ x_1]$ *elements of* 4×1 *input vector*
$rsrc_2$: $[x_2\ x_3]$ *elements of* 4×1 *input vector*

Outputs :
$rdst_1$: $[y_0\ y_1]$ (*forward/inverse*, *Hadamard*) *transformed elements of* 4×1 *output vector*
$rdst_2$: $[y_2\ y_3]$ (*forward/inverse*, *Hadamard*) *transformed elements of* 4×1 *output vector*

Attributes:

The opcode 113 and 114 are assigned to 1-D forward and inverse integer transform custom operations respectively. While, opcode 115 is assigned to 1-D Hadamard transform custom operation. The proposed custom operation for transform is also implemented using pipeline fashion and occupy two issue-slots. The other important attributes, such as opcode and latency etc., is provided in Table 7.3.

Table 7.3: Custom Operation: Transform Unit

Functional Unit	Custom Unit H264
Operation Code	113,114, 115
Number of Operands	2
Latency	2
Issue Slots	2
Modifier	None

7.3 Summary

In this chapter, we have proposed to incorporate a customized functional unit in the datapath of reconfigurable, soft-core VLIW (ρ-VEX) processor. The proposed customized functional unit implements custom operations for compute-intensive processing functions in video codec H.264/AVC. More specifically, we propose one custom operation for deblocking filter, two custom operations to compute 9 intra-prediction modes for 4×4 luma pixels-block, two custom operations for forward, inverse integer transform pair, and one custom operation to compute 4×4 Hadamard transform. Datapath for these custom opera-

tions are implemented using corresponding optimized algorithms with significantly reduced complexity as presented in previous chapters of this dissertation.

Note.

The content of this chapter is based on the the following paper:

Muhammad Nadeem, Fakhar Anjam, and Stephan Wong, *Application Specific Custom Operations for H.264/AVC on ρ-VEX: A Reconfigurable Soft-core VLIW Processor.*, <To be submitted>.

8

Experimental Results

ADVANCES in video compression standards continue to enable high-end video applications (for instance video conferencing/video calls, personal video recording, digital TV, internet video streaming, etc.) with better video quality, higher video resolutions, and lower bit-rates. The realization of advanced video coding with high-resolution videos on battery-powered mobile devices demands high complexity reduction in video coding algorithms for their real-time and low-power implementation. Consequently, the main objectives of this dissertation are to investigate how to achieve high-performance for real-time video processing applications in terms of throughput while utilizing less on-chip resources. Similarly, this dissertation also proposes adaptive, low-power hardware design for multimedia video compression applications on battery-powered electronic devices without compromising the video quality.

In previous chapters of this dissertation, we have presented several different designs for compute-intensive processing functions in H.264/AVC video codec. For experimental evaluation , the proposed design are described in VHDL and various parameters such as throughput, area-cost, and dynamic power consumption are computed using a number of typical test video sequences. . This chapter presents the experimental results and also provides a comparison of proposed hardware designs with sate-of-the-art already presented in the literature. The concluding remarks are provided at the end of this chapter.

8.1 Deblocking Filter

Deblocking filter is one of the top three compute intensive functions in H.264/AVC video codec. In Chapter 3, we have presented two hardware designs for deblocking filter. Both of these hardware designs as based on optimized deblokcing filter algorithm with significantly reduced complexity

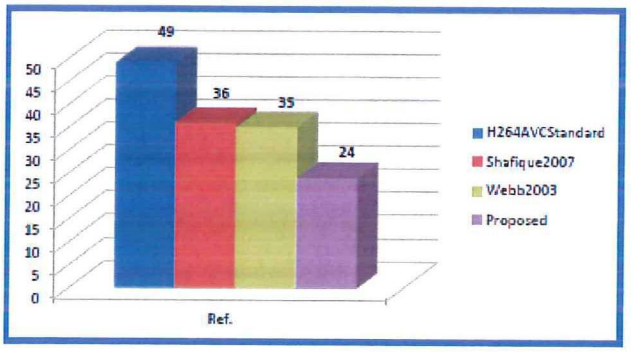


Figure 8.1: Comparison: Addition operations in strong and weak filter modes.

in terms of number of operations for its implementation. The proposed deblocking filter designs are described in VHDL and verified by comparing the RTL simulation results with those of reference software implementation. Both of these deblocking filter hardware designs are evaluated using various video test sequences and experimental results are provided in this section along with comparison with state-of-the-art presented in the literature.

8.1.1 Complexity Comparison for Deblocking Filter Algorithm

Deblocking filter is an adaptive filter and has two filter modes with varying complexity. The filter kernels for both of these modes are specified in H.264/AVC video coding standard [7], and are also provided in Algorithm 3.1. From deblocking filter algorithm perspective, we reduces the complexity of deblocking filter algorithm through novel decomposition of filter kernels and by employing various inter-filter-mode optimizations to remove redundancy in the algorithm, as described in Algorithm 3.2. The proposed optimized deblocking filter algorithm requires 24 additions and 5 clip operations for its hardware implementation. The number of additions are reduced by 51% when compared with that of original deblocking filter equations in [7], by 31% when compared with decomposition proposed in [49], and by 33% when compared with [97]. A comparison of number of operations required to carry out filter process, proposed by [7], [49], and [97] and our proposal, is depicted in Figure 8.1.

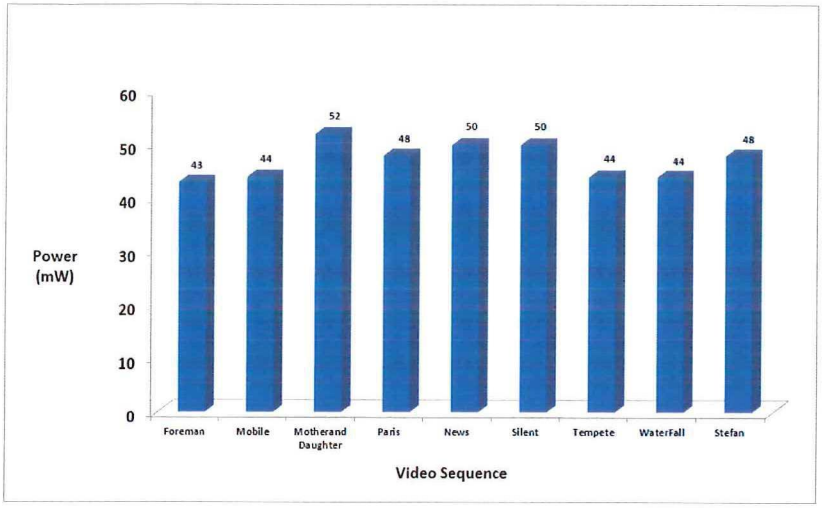


Figure 8.2: Dynamic power consumption for various test video sequences.

8.1.2 Deblocking Filter Design Using Single Filter Unit

A single-filter-unit based low-power hardware design for deblocking filter, targeting image processing applications running on battery-powered multimedia devices, is presented in Chapter 3. The low-power hardware design is described in VHDL and implementation is verified by comparing RTL simulation results for various video test sequences with those of reference software implementation [99].

The dynamic power consumption for a number of video test sequences on Xilinx Virtex II FPGA device is illustrated in Figure 8.2, while hardware-resource cost is provided in Table 8.1. A detailed comparison in terms of throughput and dynamic power consumption with other hardware designs presented in the literature is provided in Table 8.2. Since the dynamic power consumption results for deblocking filter hardware designs already presented in the literature are provided either for FPGA implementation, or for an ASIC implementation, using different video test sequences. For a fair comparison, we implemented our hardware design for both scenarios and a comparison for both implementations is provided below:

FPGA Implementation Comparison: Mustafa, et.al., [44] reported dynamic power consumption results for their deblocking filter design us-

Table 8.1: Hardware resource usage for a single deblock filter unit.

HW resource type	Available resources	Used resources	cost [%]
Slices	1408	552	39
DFF	2816	604	21
4-input LUTs	2816	662	24

Table 8.2: Throughput and dynamic power consumption comparison for deblocking filter unit.

		Parlak [44]	Thang [43]	Nadeem [17]
Filter (cyc/MB)		5248	192	192
Max Frequency	**FPGA VirtexII**	72	NA	76
(MHz)	0.18 μ**m CMOS**	200	220	200
Throughput	**FPGA VirtexII**	14	-	395
(KMB/sec)	0.18 μ**m CMOS**	38	1146	1041
Dynamic Power	**FPGA VirtexII**	85.76 mW	NA	43 mW
Consumption	0.18 μ**m CMOS**	NA	34.8 μW	16.36 μW

ing Forman video test sequence with CIF (352×288) resolution, and on 2V8000FF1157 Xilinx Virtex II FPGA device with speed grade 5. For a correct comparison, we implemented our design on the same FPGA device and computed dynamic power consumption using the same video test sequence. The test results indicate that our design consumes only 43 mW (vs. 85.76 mW). Therefore, for same video test sequence and for same FPGA device, our deblocking filter design consumes 50% less dynamic power, when compared with that of [44]. As far as throughput of hardware deblocking filter unit is concerned, with maximum operating frequency of 76 MHz and 192 cycles per MB, we can provide a maximum throughput of 395 KMB/s. Our deblocking filter design, therefore, can process full-HD (1920×1080) at 30 fps with an operating frequency as low as 59 MHz. In contrast, the deblocking filter design implementation by [44] can not provide real-time processing beyond CIF(352×288) video frame format.

ASIC Implementation Comparison: NamThang, et al., [43] reported dynamic power consumption for their deblocking filter design synthesized for 0.18 μm CMOS standard cell library. The deblocking filter core consumes 34.8 μW for processing QCIF video frame at 30 fps. We synthesized our

design using Synopsys Design Compiler (ver. v2002, rev05) under UMC 0.18μm CMOS standard cell library (v1.5). The power consumption was estimated for gate level simulation using Forman QCIF video test sequence at 30 fps. The estimated dynamic power consumption for our deblocking filter design is 16.36 μ W (vs. 34.8 μW). Our design, therefore, consumes 53% less dynamic power for the same process technology and for the same video test sequence . As far as throughput of hardware deblocking filter unit is concerned, with maximum operating frequency of 200 MHz and 192 cycles per MB, we can easily meet the real-time processing requirements of full-HD (1920×1080) video frame with an operating frequency of 59 MHz.

The dynamic power consumption of our deblocking filter hardware design is mainly reduced because of optimization of filter algorithm (more than 50% reduction of addition operations), implementation at finer granularity of processing units that can be independently deactivated during filtering process and early detection of filter-process-skip conditions to deactivate the complete processing chain.

8.1.3 Deblocking filter Design Using Dual Filter Units

For real-time video processing applications, we propose a high-throughput and area-efficient hardware design for deblocking filter in Chapter 3. The proposed design is based on dual, identical filter units and, therefore, process horizontal and vertical edges simultaneously. To achieve area-efficiency and reduce dynamic power consumption, the filter core is implemented using optimized decomposed filter kernels , already discussed in Chapter 3 of this dissertation.

For experimental evaluation purpose, the proposed deblocking filter hardware design is described in VHDL and synthesized by Synopsys Design Compiler (version v2002, rev 05), for a maximum clock frequency of 166 MHz with 0.18 μm CMOS standard cell technology (v1.5).

The maximum throughput comparison with state-of-the-art is provided in Figure 8.3. Similarly, comparison for area-cost in terms of equivalent gate count is provided in Figure 8.4. The comparison with respect to some of the other important features like SRAM , filtering cycles per MB and maximum clock frequency is also provided in Table 8.3. The last two columns in Table 8.3 provide the percentage throughput improvement and percentage area-cost reduction in terms of gate count of our proposal compared to other hardware deblocking filter designs in literature..

Figures 8.3 and 8.4, and Table 8.3 demonstrate that our deblocking filter design

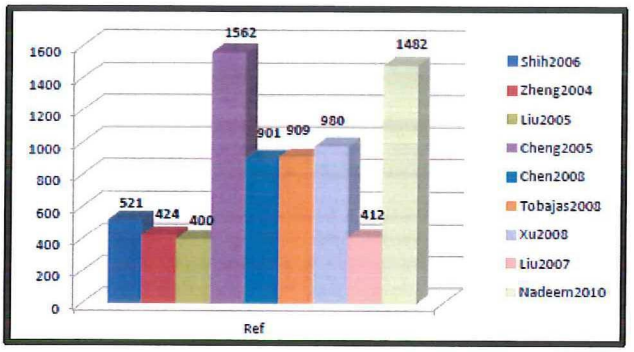


Figure 8.3: Throughput comparison for deblock filter using dual filter units

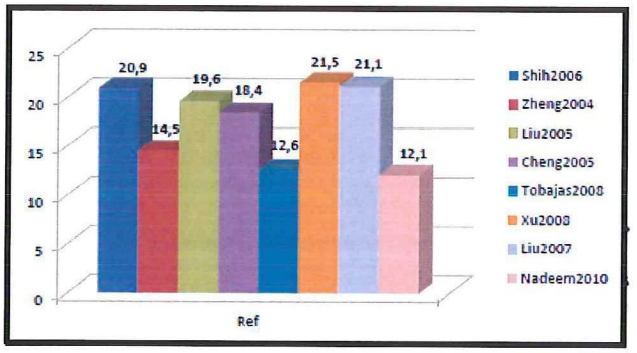


Figure 8.4: Area comparison for deblock filter using dual filter units

provides much higher throughput on one hand and requires significantly less area for its implementation on the other.

The comparisons with [48] and [27] need some clarification. The design by [48] requires almost the same area as ours, however, we provide 63% higher throughput. While on the other hand, reference [27] provides almost the same throughput as we do, however, we require 35% less area in terms of equivalent gates for its implementation.

The higher throughput of our design is achieved by merging the processing elements on the critical path and making it shorter. The area reduction, on the other hand, is mainly achieved because of:

• Algorithm level optimization - through decomposition of the filter kernels, inter-filter-mode optimizations and overlapped datapaths, we are able to reduce the number of additions by 51% and, therefore, reduce the combinatorial logic for the implementation;

• Hardware design level optimization - through efficient pipeline stage design we reduce the number of registers required for intermediate results for the next pipeline stage, reuse the same hardware resource for the realization of transpose units by identification of mutual exclusive operations in processing chain.

From the comparison with other designs presented in Table 8.3, we suggest that the proposed hardware accelerator requires 17%-44% less area in terms of equivalent gate count on one hand and provides up to 271% higher throughput on the other. The designs by [38], [31], [56], though require less area in terms of equivalent gate count. However, this is not a fair comparison as these three designs do not include the logic for the boundary strength computation, where as, our design includes the boundary strength computation unit. As far as the throughput is concerned, the proposed hardware design is 24% better when compared with [38] having highest throughput among [38], [31], [56].

8.2 Intra Prediction

In Chapter 4, we have presented a high-throughput and area-efficient hardware design for intra-prediction in H.264/AVC. The proposed design supports 4 intra-prediction modes for luma 16×16 pixels-block type, 4 intra-prediction modes for chroma 8×8 pixels-block type, and 9 intra-prediction modes for 4×4 pixels-block type in H.264/AVC. The hardware design for intra-prediction unit is based on optimized algorithm with reduced complexity

Table 8.3: Comparison for deblock filter using dual filter units

Ref	Process [μm]	Filtering [$Cycles/MB$]	Frequency [MHz]	SRAM [$bits$]	Throughput [$K-MB/s$]	Area [$K\ gates$]	Throughput Improvement[%]	Area Reduction [%]
Cheng [31]	0.18	336	100	80×32	298	9.2[a]	397	-
Chang [56]	0.18	342	100	96×32	292	11.8[b]	407	-
Li [38]	0.18	192	230	160×32	1198	9.6[c]	24	-
Shih [46]	0.18	192	100	$(128+1.5\times FW)\times32$	521	20.9	184	42
Zheng [52]	0.18	236	100	$(160+2\times FW)\times32$	424	14.5	250	17
Liu [40]	0.18	250	100	$(96+2\times FW)\times32$	400	19.6	271	39
Chen [27]	0.18	128	200	28×32	1562	18.4	-5	34
Chen [29]	0.18	222	200	64×32	901	18.7	64	35
Tobajas [48]	0.18	110	100	64×32	909	12.6	63	4
Xu [50]	0.18	204	200	$2\times96\times32,\ 2\times FW\times32$	980	21.5	51	44
Liu [41]	0.18	243	100	$2\times96\times32,\ 2\times(FW+12)\times32$	412	21.1	260	43
Nadeem [16]	0.18	112	166	64×32	1482	12.1	-	-

[a] Gate count w/o boundary strength computation
[b] Gate count w/o boundary strength computation
[c] Gate count w/o boundary strength computation

Table 8.4: Performance comparison for Intra-prediction unit.

	Intra-pred mode	Sahin [71]	Xun [65]	Wang [73]	Nadeem [18]
Luma 4x4 [cycles]	VT	17	4	17	1
	HZ	17	4	17	1
	DC	21	4	20	2
	DDL	24	4	18	1
	DDR	24	4	18	1
	VR	23	4	19	1
	HD	23	4	19	1
	VL	22	4	19	1
	HU	20	4	18	1
Luma 16x16 [cycles]	VT	-	66	17	16
	HZ	-	66	17	16
	DC	-	66	20	22
	Plane	340	66	20	22

as described in Algorithm 4.1.

For experimental evaluation , the proposed design is described in VHDL and synthesized by Synopsys Design Compiler (v2002, rev. 05) for a maximum operating frequency of 150 MHz with 0.18 μm CMOS standard cell technology (v1.5). The implementation is verified by comparing the simulation results, generated using ModelSim 6.5 for various video test sequences, with those of reference software implementation [99].

The number of cycles to compute all intra-prediction modes are provided in Table 8.4. With maximum operating frequency of 150 MHz, the proposed intra prediction unit can easily meet the real-time processing requirement of full-HD video frame resolution. The performance comparison with other state-of-the-art intra-prediction hardware designs for H.264/AVC video decoder is also provided in the same table. The comparison suggests that our design provides 50% -75% performance improvement for the luma 4×4 intra-prediction modes, when compared with [65]. For luma 16×16 case, our design provides similar performance when compared with [73].

The proposed hardware design consumes 21K gates. The area-cost of our design can not be directly compared with [71] and [73] as these designs are implemented on FPGA. The hardware design presented in [65] consumes 28.7 K gates. Since our proposed design consumes only 21 K gates, therefore, it provides 27% area saving when compared with [65].

The area-cost is reduced because of implementation of optimized intra-prediction algorithm with significantly reduced number of additions opera-

tions (27-60% reduction). Moreover, overlapped datapath for 4×4 ,8×8, and 16×16 pixels-block types also helps to greatly reduce area-cost for hardware implementation of intra-prediction unit.

8.3 Forward Integer Transform

In Chapter 5, we have presented two solutions for realization of integer transform in the processing chain of intra-frame encoder applications. The first solution targets image compression applications running on battery-powered electronic devices, such as digital still camera. The proposed solution utilizes a novel 2-D transform to derive integer-transformed coefficients directly from that of Hadamard-transformed coefficients with significantly reduced number of operations and, therefore, area and power consumption. The second solution targets real-time video compression applications using intra-frames only. The proposed solution reduces the effective latency penalty of forward integer transform to zero. Moreover, it aggressively reuses the intermediate results from Hadamard transform and, therefore, requires significantly reduced number of operations and, thus, less area for its hardware implementation

Both of the hardware designs for realization of integer transform are evaluated experimentally by describing them VHDL and synthesized using Synopsys Design Compiler (v2002 rev05) under 0.18 μm CMOS standard cell library (v1.5). The implementation for these designs, was verified, as useual, by comparing the simulation results for a set of video test sequences with those of reference software implementation [99]. The maximum operating frequency for synthesized design is 200 MHz. After logic synthesis, the dynamic power consumption was estimated using Synopsys PrimePower^{TM}. Experimental evaluation of proposed designs is provided as follows:

8.3.1 Forward Integer Transform for Image Processing Applications

The comparison of proposed hardware design for single 4×4 integer transform unit with state-of-the-art is provided in Table 8.5. From the comparison with other solutions, we suggest that the proposed solution provides the minimum latency penalty (4.82 ns) among all solutions in Table 8.5. and also requires significantly less area (2.6K gates) in terms of equivalent gate count for its hardware implementation. Therefore, it provides up to 5 times better performance in terms of throughput/area ratio for the same process technology. This

Table 8.5: Comparison for single forward integer transform unit.

Implementations	Cheng [89]	Roman [95]	Lin [92]	Nadeem [19]
DPR[a] (pixels/cycle)	8	16	8	16
Transpose / Permutation	Perm.	Perm.	Perm.	Perm.
Transform Type	4×4	4×4	4×4	4×4
Technology [μm]	0.35	0.18	0.35	0.18
Latency [ns]	10.93	6.30	30.66	4.82
Speed [MHz]	91	159	33	200
Throughput [Mpixels/s]	732	2552	261	3200
Area [gates]	2539	11727	15327	2638
Throughput / Area[KPixels/s]	288	218	17	1213
Power [mW]	NA	NA	NA	3.7[b]@31.25MHz
Throughput / Power [MPixels/s/mW]	NA	NA	NA	135

a: Data processing rate;
b: The result computed for same throughput as that of [80];

is achieved by significantly reducing number of addition operations (30 vs. 64) and higher data processing rate (16 vs. 4) for the realization of the forward integer transform. The power consumption for the other single 4×4 forward integer transform solutions is not available, therefore, we cannot compare the dynamic power consumption results.

8.3.2 Forward Integer Transform for Video Processing Applications

We have presented a design with low-latency and area-efficient realization of forward integer transform unit in the intra-frame processing chain. With this proposed solution, the effective latency penalty for forward integer transform unit is reduced to zero, as the computation for forward integer transform is no longer on the critical path. For comparison with multiple transform solutions, we merged a 4×4 Hadamard transform unit with our proposal. The comparison results for multiple transform solution are provided in Table8.6. In additions to zero latency penalty in the intra-frame processing chain, the proposed solution provides up to 30 times better performance in terms of throughput/area ratio. This is achieved by aggressively reusing the intermediate results from Hadamard transform unit to reduce the number of addition operations by 50% (32 vs. 64) and higher operating frequency (200 MHz)

Table 8.6: Comparison for multiple transform unit.

Implementations	Huang [90]	Fan [80]	Woong [94]	Nadeem [19]
DPR[a] (pixels/cycle)	8	4	16	16
Transpose / Permutation	Transpose	Transpose	Permutation	Permutation
Transform Type	Multiple	Multiple	Multiple	Multiple
Technology [μm]	0.18	0.18	0.18	0.18
Latency [ns]	20	16	5	4.92
Speed [MHz]	50	125	200	200
Throughput [Mpixels/s]	400	500	3200	3200
Area [gates]	39800	6458	63618	9926
Throughput / Area [Kpixels/s]	10	77	50	323
Power [mW]	38.7@50MHz	9.1[b]@ 62.5MHz	86.9@200MHz	78.52 @ 200MHz 4.08 @ 16MHz
Throughput / Power [MPixels/s/mW]	10.34	54.9	36.8	40.75

a: Data processing rate;
b: Power consumption by transpose register, estimated by PrimePower is 4.102 mW [94];

and data processing rate (DPR : 16 pixels/cycle). The comparison of dynamic power consumption with [80] needs some clarifications. The solution in [80] consumes 9.1 mW for a throughput of 250 M pixels / s, our proposed solution consumes 78.52 mW for a throughput of 3200 M pixels / s (approximately 13$\times$ higher throughput). However, the dynamic power consumption for the proposed solution for the same throughput as that of [80], is only 4.08 mW.

As [95], [80], [94], and [90] have been synthesized their designs for the same technology as that for our proposal, the performance and power efficiency benefits can be derived straight from Table. 8.5 and Table. 8.6 . Regarding [89] and [92] in Table. 8.5, further investigations suggest that our design is more performance and power efficient. Compared to [89], which is the more efficient design of the two, we identify that our proposal provides 2$\times$ higher data processing rate yet do not require a set of (8, 13-bit) multiplexer units to implement datapaths with and without shift operations to realize multiplications. Therefore, the proposed solution shall not only require less area for its hardware implementation for 0.35 μm process technology, but also potentially consumes low dynamic power because of reduced number of processing units.

8.4 Inverse Integer Transform

In Chapter 6, we have presented 2 hardware designs for inverse integer transform in H.264/AVC. The first design is intended for intra-frame encoder with significantly reduced latency. The proposed design processes input data on-the-fly in order to compute inverse integer transformed data block. From video processing application perspective, the second proposed hardware design for inverse integer transform, targets targets high-throughput and significantly less dynamic power consumption for its implementation. Experimental evaluation of proposed designs is provided as follows:

8.4.1 Low-latency Inverse Integer Transform Design Evaluation

The proposed low-latency inverse integer transform design is described in VHDL and synthesized by Synopsys Design Compiler (v2002, rev. 05) for a maximum operating frequency of 375 MHz with 0.18 μm CMOS standard cell technology (v1.5). The implementation is verified by comparing the simulation results for various video test sequences with those of reference software implementation [99].

The comparison of proposed hardware design for inverse integer transform unit with other state-of-the-art designs in literature is provided in Table 8.7. The proposed design requires significantly less area (7512 gates) in terms of equivalent gate count for its hardware implementation. The proposed design with a maximum operating frequency of 375 MHz, therefore, can easily meet the throughput requirement of the real-time processing of HDTV (1920×1080; 16 : 9; 30 frame per second) with an operating frequency as low as 94 MHz. It should be noted that all other designs in Table 8.7require one or more additional buffers to interface with the inverse quantization processing unit in the intra-frame processing chain. Therefore, they require additional chip-area for interface, whereas our proposed design does not require such buffers because of the fact that it processes the input data-items on-the-fly and therefore, requires only a single 16-bit register (it is included in 7512 gates).

The latency penalty of the proposed design is 2.67 ns. The latency penalty for our design can not be compared directly with those of [89], [92], and [93], because of different process technology. The scale factors for a set of different latest process technologies are provided in [100]. However, it does not provide a scale factor for 0.18 μm and 0.35 μm technologies. Since a scale factor of 2.2 exist among technologies like (80nm and 40nm), and (70nm and 36nm).

Table 8.7: Performance comparison for low-latency inverse integer transform unit

	Cheng [89]	Lin [92]	Wang [93]	Chen [87]	Woong [91]	Nadeem [21]
DPR[a] (pixels/cycle)	8	8	4	16	16	1
Transpose / Permutation	Perm	Perm	Trans	Perm	Perm	Perm
Technology (μm)	0.35	0.35	0.35	0.18	0.18	0.18
Latency (ns)	11.66	16.08	8.27	6.02	4.92	2.67
	(5.30*)	(7.31*)	(3.76*)	-	-	-
Speed (MHz)	86(189*)	56(137*)	125(266*)	166	203	375
Agg. Throughput	5.38	3.50	7.81	10.38	12.69	23.44
(4 × 4 blocks / s)	(11.81*)	(8.56*)	(16.63*)	-	-	-
Area (gates)	3377[b]	8264[b]	4424[bc]	7497[b]	5639[b]	7512

a: Data processing rate;

b: Requires an additional interface buffer (area NOT included);

c: Requires a transpose buffer (area NOT included);

*: Values computed with scale factor 2.2

Therefore, we assume the same scale factor to compare the latency penalty of our design with those of [89], [92], and [93]. The comparison shows that the proposed design provides the minimum latency penalty (2.67ns) among all designs.

The comparison Table 8.7 also provides the aggregated throughput of all of these designs of inverse integer transform unit when interfaced with Inverse Quantization (IQ) processing unit. It is assumed that the inverse quantization processing unit operates at the same maximum operating frequency as that of the corresponding design. From Table 8.7, we suggest that the proposed design provides the highest throughput among all designs. The throughput of other designs can be improved by providing 4, 8, or 16 data items in parallel. However, this approach cost additional chip-area for quantization and inverse quantization processing units with same scale factor (i.e., 4, 8, or 16).

8.4.2 Configurable, High-throughput Inverse Integer Transform Design Evaluation

A configurable, high-throughput hardware design for inverse integer transform is also introduced in Chapter 6. The proposed hardware design is based on an optimized data-driven algorithm for the inverse integer transform. For experimental evaluation, the proposed inverse integer transform design is described

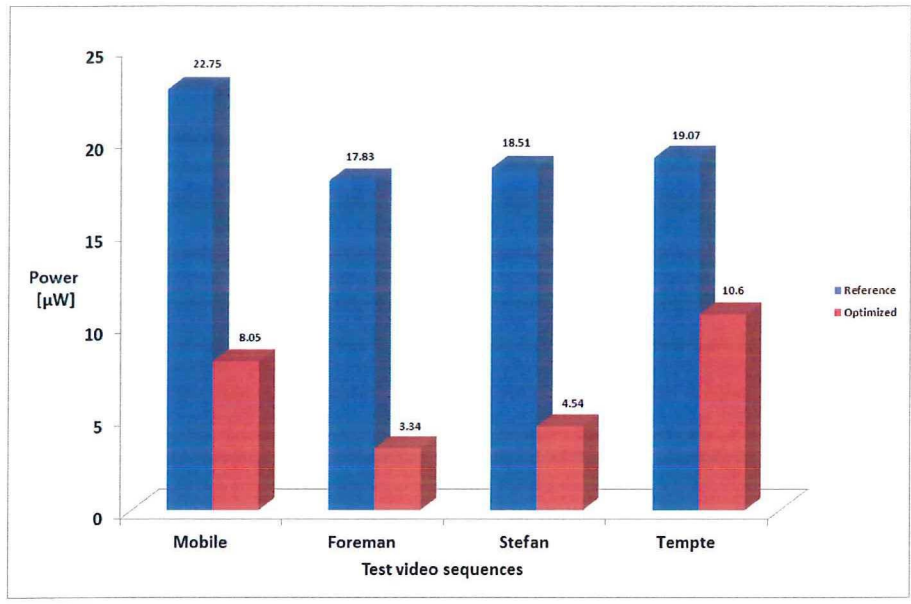
in VHDL and synthesized by Synopsys Design Compiler (v2002, rev. 05) for a maximum operating frequency of 166 MHz with 0.18 μm CMOS standard cell technology (v1.5). The implementation is verified by comparing the simulation results for various video test sequences with those of reference software implementation [99].

Since the dynamic power consumption for the inverse integer transform unit using video test sequence data is not provided in the literature, we implemented a parallel 2-D inverse integer transform based on architecture proposed in [95] as a reference for the same technology and the maximum operating frequency. The area-cost of proposed design, in terms of equivalent gate count, is 9.6 K gates. The dynamic power consumption estimated for a number of video test sequences with fine (Q_p = 16) and coarse (Q_p = 32) quantization parameter is illustrated in Figure 8.5(a) and Figure 8.5(b) respectively.

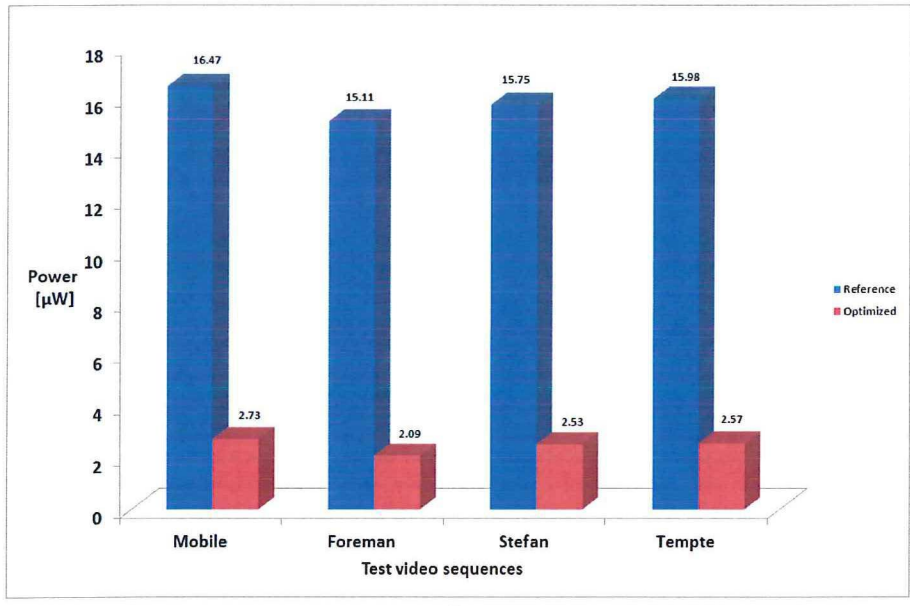
The test results suggest that dynamic power consumption is greatly affected by the Q_p value chosen and a significant reduction (up to 80%) can be achieved by using a data-driven computation algorithm for the inverse integer transform. The number of operations to compute the inverse integer transform is variable and depends on the input block type in contrast to the conventional inverse integer transform algorithm with constant number of operations for all types of input blocks. This result in a significantly reduced signal activity in the hardware circuit and, therefore, lower dynamic power consumption. The area-overhead for such an algorithm was reduced by designing a configurable processing unit to share the hardware resources.

8.5 Application Specific Custom Operations

In Chapter 7, custom operations for 3 compute-intensive processing units in H.264/AVC, are proposed for reconfigurable, soft-core, VLIW processor (ρ-VEX). The Customized Functional Unit (CFU) implements these new application specific custom operations with latency of 2 cycles each as depicted in Figure 7.1. The proposed custom operations for deblocking filter, intra-prediction and transform units are based on the optimized algorithms already provided in respective previous chapters in this dissertation. In this section, we evaluate the performance gain and also compute related hardware resource overhead for these custom operations reconfigurable, soft-core, VLIW processor (ρ-VEX).



(a)



(b)

Figure 8.5: Dynamic power consumption of inverse integer transform unit (a) Case: Qp = 16 , (b)Case: Qp = 32 .

Table 8.8: Resource utilization for deblocking filter custom operation

Resource Type	Available	ρ-VEX (Base) Used	%	ρ-VEX (Deblocking Filter) Used	%	Overhead △	%
Slices	89088	16152	18%	17100	19%	948	5.9%
FF	178176	3584	2%	3580	2%	-4	-0.1%
4-input LUT	178176	31053	17%	32845	18%	1792	5.8%

Table 8.9: Performance comparison for deblocking filter custom operation

Filter mode	ρ-VEX (Base)		ρ-VEX (Deblocking Filter)		Reduction	
	Cycles	Exec. time (msec)	Cycles	Exec. time (msec)	Cycles	%
Strong Filter	30	300	13	135.4	17	54.9%
Weak Filter	28	280	13	135.4	15	51.6%

8.5.1 Deblocking Filter Custom Operation

The deblocking filter algorithm for single edge filtering operation is extracted from reference software implementation of H.264/AVC video codec [99]. Two variants of the algorithm, with and without proposed custom operation, are implemented in C. The implementations are verified by comparing the xSTsim [101] generated simulation results with those of reference software JM 13.2 [99]. The C-code was compiled/linked using VEX tool-chain version 3.43 for 4-issue ρ-VEX processor instance with default optimization settings.

The hardware resource utilization and overhead for implementation of deblocking filter custom operations is provided in Table 8.8. Similarly, the corresponding performance gain is provided in Table 8.9.The results from Table 8.9 suggest that the strong filter mode consumes 30 cycles to compute the filtered pixels values, while 28 cycles are required to compute the filtered pixel values in the case of weak filter mode on a 4-issue ρ-VEX processor based on native ISA. With the proposed custom operation (CU_DBF_H264), the number of cycles is reduced to 13 for both filtering modes. With the incorporation of proposed custom operation in 4-issue ρ-VEX base processor, the maximum operating frequency is reduced from 99.54 MHz to 95.99 MHz. This suggests a net reduction in compute time by 55% for Strong Filter mode and 52% for the Weak Filter mode respectively after compensating the reduction in maximum operating frequency of the ρ-VEX processor. This significant reduction in compute time is at the cost of only ~6% increase in the hardware resources on Xilinx Virtex-4 (xc4vlx200-11ff1513) FPGA device.

Table 8.10: Resource utilization for intra prediction custom operation

Resource Type	Available	ρ-VEX (Base) Used	%	ρ-VEX (Intra Prediction) Used	%	Overhead △	%
Slices	89088	16152	18%	18606	20%	2454	15.2%
FF	178176	3584	2%	3992	2%	408	11.4%
4-input LUT	178176	31053	17%	35674	20%	4621	14.9%

8.5.2 Intra-Prediction Custom Operation

The H.264/AVC video coding standard provides algorithm to compute the intra predicted pixel values for a 4×4 block. An optimized version of this algorithm, with reduced number of operations, is implemented in reference software [99]. We extracted this implementation of intra-prediction unit for comparison purpose. Similar to deblocking filter case, two variants of the algorithm of intra-prediction, with and without proposed custom operations, are implemented in C. The implementations are verified by comparing the xSTsim [101] generated simulation results with those of reference software JM `13.2` [99]. The C-code was compiled/linked using VEX tool-chain version `3.43` for 4-issue ρ-VEX processor instance with default optimization settings.

The hardware resource utilization and overhead for implementation of intra prediction custom operations is provided in Table 8.10. Similarly, the corresponding performance gain is provided in Table 8.11. The compute-cycles for different intra-prediction modes vary for reference implementation. However, it remains constant for the implementation using custom operations for intra-prediction computation. The experimental results from Table 8.11 suggests that compute-time is reduced from ~40% to ~59% for various intra prediction modes with an exception of vertical intra-prediction mode. This is because of the fact that vertical intra-prediction mode do not involve any processing and, therefore, consumes similar cycles as that in the case of custom operation. The remaining intra-prediction modes, where computations are to generate predicted values clearly benefit from the intra-prediction custom operation. This significant reduction in compute-time is at the cost of around ~15% increase in the hardware resources on Xilinx Virtex-4 (xc4vlx200-11ff1513) FPGA device.

Table 8.11: Performance comparison for intra prediction custom operation

Intra Prediction mode	ρ-VEX (Base)		ρ-VEX (Intra prediction)		Reduction	
	Cycles	Exec. time (msec)	Cycles	Exec. time (msec)	Cycles	%
Vertical	10	100	11	115	-1	-0.2
Horizontal	19	190	11	115	8	39.5
DC	19	190	11	115	8	39.5
Diagonal Down Left	28	280	11	115	17	58.9
Diagonal Down Right	23	230	11	115	12	50.0
Vertical Right	21	210	11	115	10	45.2
Horizontal Down	28	280	11	115	17	58.9
Vertical Left	23	230	11	115	12	50.0
Horizontal Up	21	210	11	115	10	45.2

8.5.3 Integer/Hadamard Transform Custom Operation

The H.264/AVC compliant video encoder implementation [7] utilizes multiple transforms, such as forward and inverse integer transform and Hadamard transform operating on a 4×4 input pixels-block. A custom operation for forward and inverse integer transform along with Hadamard transform for 1-D case is proposed in Chapter 7. For performance comparison , we extracted the corresponding routines from reference software [99] and implemented the algorithm with and without custom operation for 2 scenarios. In first case, only 1-D transformation is computed. The purpose of this implementation is to determine the minimum performance gain for the transform custom operation. For the second scenario, 2-D (forward/inverse integer, Hadamard) transforms with and without utilizing the corresponding 1-D custom operation are computed. The algorithm is implemented in C. The implementations are verified by comparing the xSTsim [101] generated simulation results with those of reference software JM 13.2 [99]. Same as with previous cases, the C-code was compiled/linked using VEX tool-chain version 3.43 for 4-issue VLIW processor instance with default optimization settings.

The resource utilization for transform custom operation is provided in Table 8.12. The number of cycles and compute time for 1-D and 2-D transform implementations is provided in Table 8.13. The experimental results in Table 8.13 suggest that ~40% reduction in compute time is achieved in the case of 1-D transform computation. Similarly, for 2-D case, the compute time is reduced from ~49% to ~53% for various transforms types. The less reduction in case of Hadamard transform is observed because of the fact that the complexity of Hadamard transform is comparatively lower than that of forward and inverse integer transform cases. This significant reduction in compute time is at the

Table 8.12: Resource utilization for transform custom operation

Resource		ρ-VEX (Base)		ρ-VEX (Transform)		Overhead	
Type	Available	Used	%	Used	%	△	%
Slices	89088	16152	18%	17586	19%	1434	8.9%
FF	178176	3584	2%	3476	1%	-108	-3.0%
4-input LUT	178176	31053	17%	33834	18%	2781	9.0%

Table 8.13: Performance comparison for transform custom operation

Transform Type	ρ-VEX (Base)		ρ-VEX (Deblocking Filter)		Reduction	
	Cycles	Exec. time (msec)	Cycles	Exec. time (msec)	Cycles	%
Integer-1D	13	130	7	72.9	6	43.9
Inv Integer-1D	12	120	7	72.9	5	39.3
Hadamard-1D	12	120	7	72.9	5	39.3
Integer-2D	76	760	34	354.2	42	53.4
Inv Integer-2D	75	750	34	354.2	41	52.8
Hadamard-2D	69	690	34	354.2	35	48.7

cost of around ~9% increase in the hardware resources on Xilinx Virtex-4 (xc4vlx200-11ff1513) FPGA device.

8.6 Summary

In this chapter, we presented experimental results for the proposed designs, in terms of throughput, area and dynamic power consumption. A comparison with the designs state-of-the-art is also provided. Similarly, performance gain and the corresponding cost in terms of hardware resource-overhead for proposed custom operations is provided at the end of this chapter.

9

Conclusions and Future Directions

IN THIS this dissertation, we have presented several designs for compute-intensive processing functions in H.264/AVC. The proposed hardware designs are based on optimized algorithms with significantly less number of operations, when compared with those for original algorithms proposed in the H.264/AVC video codec standard . Similarly, custom operations for the same compute-intensive functions are also proposed in this dissertation. These custom operations are implemented in the datapath of reconfigurable, extensible, soft-core VLIW processor (ρ-VEX). The performance gain and hardware resource-overhead for these custom operations is determined experimentally and presented in this dissertation.

In this chapter, we present concluding remarks along with major achievements and possible future directions for this research work. The chapter is organized as follows: Section 9.1 provides summary of main conclusions of this dissertation. Main contributions of this dissertation are listed in Section 9.2. In Section 9.3, the problem statements addressed in this dissertation are revisited. Finally, Section 9.4 outline some open issues and future directions based on this research work.

9.1 Summary and Contributions

In **Chapter** 1, the importance of digital video coding, typical hardware platforms for video processing along with their merits and shortcomings are presented. The challenges, problem statements and methodology to achieve high-performance video processing systems are also presented in the same chapter. A motivation for a high-throughput, area-efficient and low-power solutions is described. We argued that even though processor speeds and network bandwidths continue to increase, effective video coding/compression is essential to

almost all of the multimedia consumer applications and markets.

In **Chapter** 2, we have presented an overview of the latest and state-of-the-art video coding standard H.264/AVC. Several different functional blocks of the video codec are briefly described. The compute-intensive functional units within the scope of video coding standard are identified. The state-of-the-art high performance designs for these compute-intensive functional units, already presented in the literature, are introduced briefly as part of related work. The shortcomings of these hardware designs are also highlighted. Subsequently, ρ-VEX reconfigurable, extensible, soft-core VLIW processor as a target hardware platform is introduced. The background knowledge for the processor's architecture and design is also explained at the end of this chapter in relation with our subsequent proposal for custom operations for video processing.

In **Chapter** 3, we a low-power hardware design for deblocking filter core in H.264/AVC is presented. As part of optimization of deblocking filter algorithm, a novel decomposition of the deblocking filter kernels to reduce the number of operations by more than 51% is proposed. Furthermore, we identified mutually exclusive independent processing units within the optimized algorithm and proposed to implement them separately with clock gating. . Subsequently, for real-time video processing applications, we have presented a high-throughput, area-efficient, hardware accelerator for the deblocking filter in H.264/AVC. The optimization techniques employed and proposed hardware design provides significantly higher throughput and reduced the on-chip area. The proposed design easily provide real-time filtering operation for the HDTV video format (4096×2304, 16:9) at 30 fps and meet the throughput requirements of all levels (level 1-5.1) in H.264/AVC video coding standard.

Similarly, in **Chapter** 4, a high-performance hardware design for intra-prediction in H.264/AVC is presented. Again, in order to reduce the complexity of the intra-prediction algorithm, optimizations are carried out to lower the number of operations. The optimization of algorithm for 4×4 intra-prediction modes enabled us to reduce the number of addition by 60% when compared with that of standard prediction equations. Similarly, overlapped data-paths for the 4×4 ,8×8, and 16×16 block types also helped to greatly reduce the required on-chip resources for its hardware implementation. With an operating frequency of 150 MHz, the proposed intra-prediction unit can easily meet the real-time processing requirement of HDTV video frame formats.

In **Chapter** 5, we proposed two solutions for realization of the forward integer transform in the processing chain of intra-only frame encoder applications. The first solution targeted image compression applications running on

battery-powered electronic devices, such as Digital Still Camera (DSC). The proposed solution utilized a novel 2-D transform to derive the forward integer transform coefficients directly from that of Hadamard transform with significantly reduced number of operations and, therefore, area and power consumption. The second solution targeted video compression applications processing high-resolution video frames in real-time, such as Digital Video Camera (DVC), Television Studio Broadcast and Surveillance video. The proposed solution reduced the effective latency penalty of the forward integer transform to zero. Moreover, it aggressively re-used the intermediate results from Hadamard transform and, therefore, requires significantly reduced number of operations and thus less area for its hardware implementation.

In **Chapter** 6, we proposed two hardware designs for the inverse integer transform in H.264/AVC video codec. The first design is intended for intra-frame encoder with reduced latency. The proposed design process the input data on-the-fly to produce the inverse transformed data block. A data-driven algorithm with variable number of operations is also introduced in this chapter. The second hardware design, based on the data-driven algorithm, enabled us to provide high-throughput and consume significantly less dynamic power for its implementation.

In **Chapter** 7, we proposed to incorporate a customized functional unit in the datapath of reconfigurable, soft-core, VLIW processor. This customized functional unit implements application specific custom operations for the compute intensive processing functions in video codec H.264/AVC. More specifically, one custom instruction for deblocking filter, two custom operationss to compute the complete predicted 4×4 pixel block in intra-prediction module and another three custom operationss for forward/inverse integer transform along with Hadamard transform are proposed in this chapter. It is proposed to use the optimized algorithms with significantly reduced number of operation for the implementation of these custom operations in CFU.

In **Chapter**8, we have presented experimental results for the proposed design in previous chapters of this dissertation. A comparison with state-of-the-art is also presented in the same chapter.

9.2 Main Contributions

The research carried out in the course of this PhD project is published in several scientific publications. This section highlights the main contributions of the research work described in this dissertation and are provided as follows:

1. In-loop deblocking filter.

    - The complexity of in-loop deblocking filter algorithm is reduced by novel decomposition of filter kernels and intra-module optimizations. The complexity of the deblocking filter algorithm is reduced by more than 51% when compared with that of algorithm described in the video coding standard H.264/AVC [Section: 3.2], [16].
    - A low-power hardware design for deblocking filter unit is proposed in this dissertation. The proposed design consumes 43 mW dynamic power on a Xilinx Virtex II FPGA and consumes 16.36 μW, when synthesized using 0.18 μm CMOS standard cell library. The FPGA implementation on Virtex II can work at 76 MHz, whereas the maximum operating frequency for 0.18 μm process technology is 200 MHz. Experimental results suggest that the dynamic power consumption is reduced up to 50%, when compared with state-of-the-art designs in literature. [Table:8.2] [17].
    - For real-time video processing applications, a high-throughput, area-efficient hardware design, based on the low-power filter unit, is proposed. . This design utilizes 2 filter units and, therefore, can process input pixels on-the-fly in both horizontal and vertical directions. The proposed design provides significantly higher throughput (more than 63% when compared with state-of-the-art in literature having similar area-cost) and require less on-chip area (around 35% when compared with state-of-the-art in literature having similar throughput) [Table: 8.3] [16].

2. Intra-prediction

    - The complexity of intra-prediction algorithm for various intra-prediction modes is reduced by 27% - 60% in comparison with intra-prediction algorithm proposed in H.264/AVC video coding standard. A configurable, high-throughput, and area-efficient hardware design for intra-prediction unit, based on the reduced complexity intra-prediction algorithm, is proposed in this dissertation. The comparison with other state-of-the-art suggests that our proposed hardware design provides 50%-75% performance improvement and requires only 21K gates for its implementation, when synthesized under 0.18 μm CMOS standard cell technology [18].

3. Forward integer transform

   - A transformation, to compute integer-transformed residual block from Hadamard-transformed residual block in the processing chain of H.264/AVC video codec, is proposed in this dissertation. This approach reduces the number of addition operations by more than 50% for realization of integer transform in the video codec. The comparison with the existing single 4×4 forward integer transform solutions from the literature suggest that the proposed solution provides the minimum latency penalty (4.82ns) among all solutions and also requires significantly less area (2.6K gates) in terms of equivalent gate count for its hardware implementation. Therefore, it provides up to 5 times better performance in terms of throughput/area ratio for the same process technology. [19].
   - Similarly, a low-latency and area-efficient solution for realization of forward integer transform unit in the intra-frame processing chain is proposed. With this proposed solution, the effective latency penalty for the forward integer transform unit is reduced to zero. In additions to zero latency penalty in the intra-frame processing chain, the proposed solution provides up to 30 times better performance in terms of throughput/area ratio. [19].

4. Inverse integer transform

   - For inverse integer transform unit, a configurable, low-power hardware design is presented in this dissertation. The proposed design is based on a data-driven computation algorithm for the inverse integer transform. It efficiently exploits the zero-valued coefficients in the input blocks to reduce dynamic power consumption. The experimental results show that the proposed design consumes significantly less dynamic power (up to 80% reduction), when compared with existing conventional designs for the inverse integer transform, with a small area-overhead (approximately 2 K gates) [20] [21].

5. Custom instructions/processing units for extensible, reconfigurable, soft-core VLIW processor (ρ-VEX).

   - In this dissertation, we also proposed custom operations for deblocking filter, intra-prediction and forward/inverse integer transform units in H.264/AVC foe a reconfigurable VLIW processor.

The datapath is based on the optimized algorithms presented in this dissertation. The proposed custom operations significantly reduce the compute time (approximately 40% - 59%) for the corresponding compute-intensive functions in H.264/AVC on a reconfigurable, soft-core VLIW processor (ρ-VEX). [8.9, 8.11, 8.13].

9.3 Problem Statements Revisited

The answers to the research questions presented in Section 1.3 can, therefore, be summarized as follows:

- *How to achieve high-performance and flexible processing in video processing/compression applications?* High-performance in terms of high-throughput typically comes from parallelism in application kernels. Similarly, beside high-performance in terms of throughput, the realization of advanced video coding on battery-powered multimedia electronic devices with reduced on-chip area and lower dynamic power, is also extremely important. Although parallel processing or high-throughput, typically in a module for filtering process of video data, can be achieved by duplicating the circuit. However, it doesn't help in other dimensions of constraint based design-space of that module. In this dissertation, we have demonstrated that an optimized algorithm with reduced complexity is a key to achieve high-performance in terms of throughput, area and dynamic power consumption at the same time [Chapters: 3, 4, 5, 6]. The flexibility to adapt to video coding standard evolutions and market/technology induced changes can be satisfied by utilizing reconfigurable fabric. Reconfigurable processors provide the flexibility as well as performance for the applications that are amenable to acceleration with reconfigurable hardware. In this dissertation, we also proposed to extend the ISA of extensible, reconfigurable, soft-core VLIW processor (ρ-VEX) with application specific custom operations for various video processing units in H.264/AVC video coding standard. This includes deblocking filter, intra-prediction and forward/inverse integer transform units as well as Hadamard transform [7].

- *How to reduce the coding complexity and improve the real-time performance without compromising the video quality?* The video coding algorithm defines a detailed implementation outline of a required original function and, therefore, determines how to solve the problem and

how to reduce the original complexity. In order to do an optimal implementation, it is essential to fully understand the principles behind, and algorithms employed, in video coding as it is a key to reduce power consumption and improve efficiency. In this dissertation, we demonstrated that improved throughput and lower dynamic power consumption can be achieved at the same time by reducing algorithm complexity. (please refer to Section 9.2 for summary of our contributions and supporting experimental results). The optimized algorithm with lower number of operations not only require less on-chip area for its implementation, the target throughput can be achieved at lower clock rate or in less number of cycles. The optimized algorithm with reduced number of operations also helps to lower the dynamic power consumption of the processing modules. The complexity of the algorithm can be reduced by intra-module optimization to remove the redundancy in the algorithm itself (e.g. deblocking filter and intra-prediction modules), or by inter-module optimizations by reusing the previous results (e.g. forward integer transform). Similarly, the complexity can also be reduced by considering the statistical nature of the input signal in the processing module (e.g. inverse integer transform module). An adaptive design to exploit the statistical redundancy in the video signal not only helps to carry out required processing in reduced number of operations but also helps to significantly lower the the dynamic power consumption.

- *How can reconfigurable computing be utilized to improve the flexibility and performance for video coding signal processing applications?* Customization of reconfigurable processors is a way forward to improve the performance for a moderate cost. In this dissertation, we extended the ISA of ρ-VEX reconfigurable soft-core VLIW processor with application specific custom operations such as adaptive in-loop deblocking filter, intra-prediction, forward integer transform, inverse integer transform and Hadamard transform in H.264/AVC video codec. The customized modules implemented in the datapath of ρ-VEX processor, for the proposed custom operations, are based on the optimized algorithms for these modules already discussed in the dissertation. Because of the reduced number of operations and extensive parallel processing in the proposed custom operations for video processing, it not only significantly reduces the required number of cycles at the cost of slight increase in hardware resources but can potentially reduce the dynamic power consumption as well (please refer to Section 9.2 for summary of our contributions and supporting experimental results).

9.4 Future Directions

This section provides future research directions and improvements to the work presented in this dissertation:

This dissertation focused on several compute-intensive processing functions in H.264/AVC video codec and proposed low-power, area-efficient and high-throughput hardware designs. The algorithms for these compute-intensive functions were optimized to reduce algorithm complexity in terms of number of arithmetic operations for their implementation. Similar efforts can be made for video pre-, and post-processing chain. Moreover, the proposed concepts can be further extended for the image processing algorithms and also for the upcoming video coding standard H.265.

In this dissertation, we targeted to reduce the power consumption at module/algorithm level. However, there is a need to deal with the power-related issues at all abstraction levels (process technology, module/processor design, and application levels).

Custom operations for reconfigurable soft-core processor were proposed in this dissertation. These custom operations and datapaths were designed manually, which is time consuming. This process need to be automated by designing new tools similar to that used in ASIP. Similarly, low-power optimizations, intelligent algorithms and transformation techniques need to be investigated for automated design of efficient datapaths.

In Chapter 3, we proposed a hardware design for deblocking filter based on double filter units. Two edges (horizontal and vertical edges) are processed simultaneously and, the proposed design can meet the throughput requirement for H.264/AVC video codec up to level 5.1. For even higher throughput, The proposed design can be extended by employing more filter units to process all horizontal/vertical edges at 4×4, 8×4 or 16×4 pixels block level to reduce the number of filtering cycles per macroblock at the cost of additional on-chip area.

In Chapter 4, a design for intra-prediction module is presented. In video codec, the MB is either encoded in intra or inter-encoding mode. Both of these modes are mutually exclusive. Moreover, the structure of intra-prediction filter kernels and SAD computation are similar. Therefore, the same hardware resources can be efficiently utilized by reusing them for SAD computation in case of inter encoding mode of a MB.

In this dissertation, we proposed hardware designs for different compute-

intensive modules in H.264/AVC. We plan to design other modules such as CAVLC/CABAC and integrate them all to implement complete video codec.

Bibliography

[1] J. Meehan, S. Busch, J. Noel, and F. Noraz. Multimedia IP Architecture Trends in the Mobile Multimedia Consumer Device. *Image Communication.*, 25(5):317–324, June 2010.

[2] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, july 2003.

[3] Yen-Kuang Chen and S. Y. Kung. Trend and Challenge on System-on-a-Chip Designs. *Journal of Signal Processing and System*, 53(1-2):217–229, November 2008.

[4] Yen-Kuang Chen, Eric Q. Li, Xiaosong Zhou, and Steven Ge. Implementation of H.264 Encoder and Decoder on Personal Computers. *Journal of Visual Communication and Image Representation*, 17(2):509 – 532, 2006.

[5] S. Donggyu. High Efficiency Video Coding(HEVC). url-http://goo.gl/U29ql1.

[6] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update,2012-2117. `http://goo.gl/04Z3VW`, 2013.

[7] Advanced Video Coding for Generic Audiovisual Services, 2005.

[8] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video Coding with H.264/AVC:Ttools, Performance, and Complexity. *IEEE Magazine on Circuits and Systems*, 4(1):7 – 28, quarter 2004.

[9] K.; Salmimaa M.; Hallapuro A.; Lainema J. Willner, K.; Ugur. Mobile 3D Video Using MVC and N800 Internet Tablet. In *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*, pages 69–72, 2008.

[10] U.J. Kapasi, S. Rixner, W.J. Dally, B. Khailany, J.H. Ahn, P. Mattson, and J.D. Owens. Programmable Stream Processors. *Computer*, 36(8):54–62, Aug. 2003.

[11] Shuai Hu, Zhe Zhang, Mengsu Zhang, and Tao Sheng. Optimization of Memory Allocation for H.264 Video Decoder on Digital Signal Processors. In *Image and Signal Processing, 2008. CISP '08. Congress on*, volume 2, pages 71–75, May.

[12] G. R. Stewart. Implementing Video Compression Algorithms on Reconfigurable Devices, June. 2009.

[13] K. Diefendorff and P.K. Dubey. How multimedia workloads will change processor design. *Computer*.

[14] P. Faraboschi, G. Brown, J. Fisher, G. Desoll, and F. Homewood. Lx: A Technology Platform for Customizable VLIW Embedded Processing. In *International Symposium on Computer Architecture*, pages 203–213, June.

[15] P. Faraboschi J. Fisher and C. Young. Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools., 2004.

[16] M. Nadeem, S. Wong, G. Kuzmanov, and A. Shabbir. A High-throughput, Area-efficient Hardware Accelerator for Adaptive Deblocking Filter in H.264/AVC. In *IEEE/ACM/IFIP 7th Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, pages 18–27, Oct. 2009.

[17] M. Nadeem, S. Wong, G. Kuzmanov, A. Shabbir, M.F. Nadeem, and F. Anjam. Low-power, High-throughput Deblocking Filter for H.264/AVC. In *International Symposium on System on Chip (SoC)*, pages 93–98, Sept. 2010.

[18] M. Nadeem, S. Wong, and G. Kuzmanov. An Efficient Hardware Design for Intra-prediction in H.264/AVC Decoder, year=2011, month=Apr, pages=1–6,. In *Saudi International Electronics, Communications and Photonics Conference (SIECPC)*.

[19] M. Nadeem, S. Wong, and G. Kuzmanov. An Efficient Realization of Forward Integer Transform in H.264/AVC Intra-frame Encoder. In *International Conference on Embedded Computer Systems (SAMOS)*, pages 71–78, july 2010.

[20] Muhammad Nadeem, Stephan Wong, and Georgi Kuzmanov. Configurable, Low-power Design for Inverse Integer Transform in H.264/AVC. In *International Conference on Frontiers of Information Technology (IFIT)*, pages 32:1–32:5. ACM, 2010.

[21] M. Nadeem, S. Wong, and G. Kuzmanov. Inverse Integer Transform in H.264/AVC Intra-frame Encoder. In *IEEE International Symposium on Electronic Design, Test and Application (DELTA)*, pages 228–233, Jan. 2011.

[22] H.S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky. Low-complexity Transform and Quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):598 – 603, July 2003.

[23] D. Marpe, H. Schwarz, and T. Wiegand. Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, July.

[24] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz. Adaptive deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):614–619, July 2003.

[25] P. Benzie, J. Watson, P. Surman, I. Rakkolainen, K. Hopf, H. Urey, V. Sainov, and C. von Kopylow. A Survey of 3DTV Displays: Techniques and Technologies. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(11):1647 –1658, nov. 2007.

[26] Bin Sheng, Wen Gao, and Di Wu. An Implemented Architecture of Deblocking Filter for H.264/AVC. In *International Conference on Image Processing (ICIP)*, volume 1, pages 665–668, Oct 2004.

[27] Chung-Ming Chen and Chung-Ho Chen. A Memory Efficient Architecture for Deblocking Filter in H.264 Using Vertical Processing Order. In *International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 361–366, Dec. 2005.

[28] Chung-Ming Chen and Chung-Ho Chen. Configurable VLSI Architecture for Deblocking Filter in H.264/AVC. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(8):1072–1082, Aug. 2008.

[29] Qing Chen, Wei Zheng, Jian Fang, Kai Luo, Bing Shi, Ming Zhang, and Xianmin Zhang. A Pipelined Hardware Architecture of Deblocking Filter in H.264/AVC, booktitle = International Conference on Communications and Networking in China (ChinaCom), year = 2008, pages = 815—819, month = Aug.,.

[30] C. M. Cheng and C. H. Chen. A Memory Efficient Architecture for Deblocking Filter in H.264 Using Vertical Processing Order. In *IEEE International Conference on Intelligent Sensors, Sensor Networks. Information Process*, pages 361–366, 2005.

[31] Chao-Chung Cheng and Tian-Sheuan Chang. An Hardware Efficient Deblocking Filter for H.264/AVC. In *International Conference on Consumer Electronics, ICCE. 2005 Digest of Technical Papers*, pages 235–236, Jan. 2005.

[32] Chao-Chung Cheng, Tian-Sheuan Chang, and Kun-Bin Lee. An In-place Architecture for the Deblocking Filter in H.264/AVC, journal = IEEE Transactions on Circuits and Systems II: Express Briefs,, year = 2006, volume = 53, pages = 530–534, number = 7, month = July, issn = 1549-7747.

[33] Heng-Yao Lin, Jwu-Jin Yang, Bin-Da Liu, and Jar-Ferr Yang. Efficient Deblocking Filter Architecture for H.264 Video Coders. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, page 4 pp, May 2006.

[34] G. Khurana, A.A. Kassim, Tien Ping Chua, and M.B. Mi. A Pipelined Hardware Implementation of In-loop Deblocking Filter in H.264/AVC. *IEEE Transactions on Consumer Electronics*, 52(2):536–540, May 2006.

[35] Sung Deuk Kim, Jaeyoun Yi, Hyun Mun Kim, and Jong Beom Ra. A Deblocking Filter With Two Separate Modes in Block-based Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(1):156–60, Feb 1999.

[36] Byung-Joo Kim, Jae-Il Koo, Min-Cheol Hong, and Seongsoo Lee. Low-Power H.264 Deblocking Filter Algorithm and Its SoC Implementation. In Long-Wen Chang and Wen-Nung Lie, editors, *Advances in Image and Video Technology*, volume 4319 of *Lecture Notes in Computer Science*, pages 771–779. Springer Berlin / Heidelberg, 2006.

[37] Kyeong-Yuk Min and Jong-Wha Chong. A Memory and Performance Optimized Architecture of Deblocking Filter in H.264/AVC. In *International Conference on Multimedia and Ubiquitous Engineering (MUE)*, pages 220–225, April 2007.

[38] S.Goto L. Li and and T. Ikenaga. A Highly Parallel Architecture for Deblocking Filter in H.264/AVC. *IEICE Transactions on Information and Systems*, E88-D(7):1623 –1629, 2005.

[39] Lingfeng Li, Satoshi Goto, and Takeshi Ikenaga. An Efficient Deblocking Filter Architecture with 2-dimensional Parallel Memory for

H.264/AVC. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 623–626. ACM, 2005.

[40] Tsu-Ming Liu, Wen-Ping Lee, Ting-An Lin, and Chen-Yi Lee. A Memory-efficient Deblocking Filter for H.264/AVC Video Coding. In *IEEE International Symposium on Circuits and Systems, ISCAS 2005*, volume 3, pages 2140–2143, May 2005.

[41] Tsu-Ming Liu, Wen-Ping Lee, and Chen-Yi Lee. An In/Post-Loop Deblocking Filter With Hybrid Filtering Schedule. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(7):937–943, July 2007.

[42] Miao Sima, Yuanhua Zhou, and Wei Zhang. An Efficient Architecture for Adaptive Deblocking Filter of H.264/AVC Video Coding. *IEEE Transactions on Consumer Electronics*, 50(1):292–296, Feb 2004.

[43] NamThang Ta, JinSeon Youn, HuiGon Kim, JunRim Choi, and Seung-Soo Han. Low-power High-throughput Deblocking Filter Architecture for H.264/AVC. In *International Conference on Electronic Computer Technology*, pages 627–631, Feb. 2009.

[44] M. Parlak and I. Hamzaoglu. A Low Power Implementation of H.264 Adaptive Deblocking Filter Algorithm. In *NASA/ESA Conference on Adaptive Hardware and Systems*, pages 127–133, Aug. 2007.

[45] M. Parlak and I. Hamzaoglu. Low power H.264 deblocking Filter Hardware Implementations. *IEEE Transactions on Consumer Electronics*, 54(2):808–816, May 2008.

[46] Shen-Yu Shih, Cheng-Ru Chang, and Youn-Long Lin. A Near Optimal Deblocking Filter for H.264 Advanced Video Coding. In *Asia and South Pacific Conference on Design Automation*, pages 170–175., jan. 2006.

[47] W.T. Staehler, E.A. Berriel, A.A. Susin, and S. Bampi. Architecture of an HDTV Intraframe Predictor for a H.264 Decoder. In *International Conference on Very Large Scale Integration (IFIP)*, pages 228–233, Oct. 2006.

[48] F. Tobajas, G.M. Callico, P.A. Perez, V. de Armas, and R. Sarmiento. An Efficient Double-Filter Hardware Architecture for H.264/AVC Deblocking Filtering. *IEEE Transactions on Consumer Electronics*, 54(1):131–139, Feb. 2008.

[49] J. Webb. Video deblocking filter, Feb 2003.

[50] Ke Xu and Chiu-Sing Choy. A Five-Stage Pipeline, 204 Cycles/MB, Single-Port SRAM-Based Deblocking Filter for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(3):363–374, Mar 2008.

[51] Yu-Wen Huang, To-Wei Chen, Bing-Yu Hsieh, Tu-Chih Wang, Te-Hao Chang, and Liang-Gee Chen. Architecture Design for Deblocking Filter in H.264/JVT/AVC. In *International Conference on Multimedia and Expo (ICME)*, volume 1, pages I–693–6, July 2003.

[52] G. Zheng and L. Yu. An Efficient Architecture Design for Deblocking Loop Filte. In *Picture Coding Symposium*, 2004.

[53] S. Krishnan V. Venkatraman and N. Ling. Architecture for De-blocking Filter in H.264. In *Picture Coding Symposium*, 2004.

[54] K.K. Pang and T.K. Tan. Optimum Loop Filter in Hybrid Coders. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(2):158–167, Apr 1994.

[55] Yung-Lyul Lee and HyunWook Park. Loop Filtering and Post-filtering for Low-bit-rates Moving Picture Coding, journal = Signal Processing: Image Communication, year = 2001, volume = 16, pages = 871–890, number = 9, doi = 10.1016/S0923-5965(00)00048-5, issn = 0923-5965, url = http://www.sciencedirect.com/science/article/pii/S0923596500000485.

[56] Shih-Chien Chang, Wen-Hsiao Peng, Shih-Hao Wang, and Tihao Chiang. A Platform Based Bbus-interleaved Architecture for De-blocking Filter in H.264/MPEG-4 AVC, journal = IEEE Transactions on Consumer Electronics, year = 2005, volume = 51, pages = 249–255, number = 1, month = Feb., doi = 10.1109/TCE.2005.1405728, issn = 0098-3063.

[57] Fangwen Fu, Xinggang Lin, and Lidong Xu. Fast Intra Prediction Algorithm in H.264-AVC, year=2004, month=Aug, volume=2, pages=1191–1194 volume = 2,. In *International Conference on Signal Processing (ICSP)*.

[58] Chao-Chung Cheng and Tian-Sheuan Chang. Fast Three Step Intra Prediction Algorithm for 4x4 Blocks in H.264. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 1509–512, May 2005.

[59] Chao-Hsuing Tseng, Hung-Ming Wang, and Jar-Ferr Yang. Enhanced Intra-4 x4 Mode Decision for H.264/AVC Coders. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(8):1027–1032, Aug 2006.

[60] Jhing-Fa Wang, Jia-Ching Wang, Jang-Ting Chen, An-Chao Tsai, and A. Paul. A Novel Fast Algorithm for Intra Mode Decision in H.264/AVC Encoders. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, page 4 pp., May 2006.

[61] Jia-Ching Wang, Jhing-Fa Wang, Jar-Ferr Yang, and Jang-Ting Chen. A Fast Mode Decision Algorithm and Its VLSI Design for H.264/AVC Intra-Prediction. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(10):1414–1422, Oct 2007.

[62] Byeongdu La, Minyoung Eom, and Yoonsik Choe. Fast Mode Decision for Intra Prediction in H.264/AVC Encoder. In *IEEE International Conference on Image Processing (ICIP)*, volume 5, pages 321–324, Sept 2007.

[63] An-Chao Tsai, A. Paul, Jia-Ching Wang, and Jhing-Fa Wang. Intensity Gradient Technique for Efficient Intra-Prediction in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(5):694–698, May 2008.

[64] Yan Ye and M. Karczewicz. Improved h.264 intra coding based on bi-directional intra prediction, directional transform, and adaptive coefficient scanning. In *IEEE International Conference on Image Processing (ICIP)*, pages 2116–2119, Oct 2008.

[65] Xun He, Dajiang Zhou, Jinjia Zhou, and S. Goto. A New Architecture for High Performance Intra Prediction in H.264 Decoder. In *International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pages 41–44, Jan. 2009.

[66] Yu-Wen Huang, Bing-Yu Hsieh, Tung-Chien Chen, and Liang-Gee Chen. Analysis, Fast Algorithm, and VLSI Architecture Design for

H.264/AVC Intra Frame Coder. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(3):378 – 401, March 2005.

[67] Genhua Jin and Hyuk-Jae Lee. A Parallel and Pipelined Execution of H.264/AVC Intra Prediction. In *IEEE International Conference on Computer and Information Technology (CIT)*, page 246, Sept. 2006.

[68] Chun-Wei Ku, Chao-Chung Cheng, Guo-Shiuan Yu, Min-Chi Tsai, and Tian-Sheuan Chang. A High-Definition H.264/AVC Intra-Frame Codec IP for Digital Video and Still Camera Applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(8):917–928, Aug. 2006.

[69] D. Marpe, V. George, H. L. cycon, and K. U. Barthel. Performance Evaluation of Motion-JPEG2000 in Comparison with H.264/AVC Operated in Pure Intra Coding Mode. volume 13, pages 129–137, 2003.

[70] E. Sahin and I. Hamzaoglu. An Efficient Hardware Architecture for H.264 Intra Prediction Algorithm. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, April 2007.

[71] E. Sahin and I. Hamzaoglu. An Efficient Intra Prediction Hardware Architecture for H.264 Video Decoding. In *Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD)*, pages 448–454, Aug. 2007.

[72] M. Shafique, L. Bauer, and J. Henkel. A Parallel Approach for High Performance Hardware Design of Intra Prediction in H.264/AVC Video Codec. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1434–1439, April 2009.

[73] Xi Wang, Xiaoxin Cui, and Dunshan Yu. A parallel Intra prediction Architecture for H.264 Video Decoding. In *IEEE International Conference on ASIC (ASICON)*, pages 859–862, Oct. 2009.

[74] G. Raja, S. Khan, and M.J. Mirza. VLSI Architecture and Implementation of H.264 Integer Transform. In *International Conference on Microelectronics (ICM)*, pages 218–223, Dec 2005.

[75] Hanli Wang, Sam Kwong, and Chi-Wah Kok. Efficient Prediction Algorithm of Integer DCT Coefficients for H.264/AVC Optimization. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(4):547–552, April 2006.

[76] Ji Xiuhua, Zhang Caiming, and Wang Yanling. Fast Algorithm of the 2-D 4x4 Inverse Integer Transform for H.264/AVC. In *International Conference on Innovative Computing, Information and Control*, pages 144–144, Sept 2007.

[77] N. T. Ngo, T. T T Do, T.M. Le, Y. S. Kadam, and A. Bermak. ASIP-controlled Inverse Integer Transform for H.264/AVC Compression. In *IEEE/IFIP International Symposium on Rapid System Prototyping*, pages 158–164, June 2008.

[78] T.T.T. Do and T.M. Le. High throughput Area-efficient SoC-based Forward/Inverse Integer Transforms for H.264/AVC. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4113–4116, May 2010.

[79] Chih-Peng Fan. Fast 2-dimensional 4x4 Forward Integer Transform Implementation for H.264/AVC. *IEEE Transactions on Circuits and Systems*, 53(3):174–177, March 2006.

[80] Chih-Peng Fan. Cost-Effective Hardware Sharing Architectures of Fast 8x8 and 4x4 Integer Transforms for H.264/AVC. In *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pages 776–779, Dec. 2006.

[81] Chih-Peng Fan and Yu-Lian Lin. Implementations of Low-Cost Hardware Sharing Architectures for Fast 8x8 and 4x4 Integer Transforms in H.264/AVC. *IEICE Transactions on Fundam. Electron. Commun. Comput. Sci.*, E90-A(2):511–516, February 2007.

[82] Liu Ling-zhi, Qiu Lin, Rong Meng-tian, and Jiang Li. A 2-D Forward/Inverse Integer Transform Processor of H.264 Based on Highly-parallel Architecture. In *IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 158–161, July 2004.

[83] Yi-Chih Chao, Hui-Hsien Tsai, Yu-Hsiu Lin, Jar-Ferr Yang, and Bin-Da Liu. A Novel Design for Computation of All Transforms in H.264/AVC Decoders. In *IEEE International Conference on Multimedia and Expo.*, pages 1914–1917, July 2007.

[84] Bing Shi, Wei Zheng, Dongxiao Li, and Ming Zhang. Fast Algorithm and Architecture Design for H.264/AVC Multiple Transforms. In *IEEE International Conference on Multimedia and Expo*, pages 2086–2089, July 2007.

[85] Grzegorz Pastuszak. Transforms and Quantization in the High-Throughput H.264/AVC Encoder Based on Advanced Mode Selection. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 203–208, April 2008.

[86] Honggang Qi, Qingming Huang, and Wen Gao. A Low-Cost Very Large Scale Integration Architecture for Multistandard Inverse Transform. *IEEE Transactions on Circuits and Systems*, 57(7):551–555, July 2010.

[87] Kuan-Hung Chen, Jiun-In Guo, Kuo-Chuan Chao, Jinn-Shyan Wang, and Yuan-Sun Chu. A High-performance Low Power Direct 2-D Transform Coding IP Design for MPEG-4 AVC/H.264 With a SwitchingP-power Suppression Technique. In *IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test (VLSI-TSA-DAT)*, pages 291–294, April 2005.

[88] Kuan-Hung Chen, Jiun-In Guo, and Jinn-Shyan Wang. A High-performance Direct 2-D Transform Coding IP Design for MPEG-4 AVC/H.264. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(4):472– 483, April 2006.

[89] Zhan-Yuan Cheng, Che-Hong Chen, Bin-Da Liu, and Jar-Ferr Yang. High throughput 2-D Transform Architectures for H.264 Advanced Video Coders. In *IEEE Asia-Pacific Conference on Circuits and Systems*, volume 2, pages 1141–1144, dec. 2004.

[90] Chong-Yu Huang, Lien-Fei Chen, and Yeong-Kang Lai. A High-speed 2-D Transform Architecture with Unique Kernel for Multi-standard Video Applications. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 21–24, May 2008.

[91] Woong Hwangbo, Jaemoon Kim, and Chong-Min Kyung. A High-Performance 2-D Inverse Transform Architecture for the H.264/AVC Decoder. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1613 –1616, May 2007.

[92] Heng-Yao Lin, Yi-Chih Chao, Che-Hong Chen, Bin-Da Liu, and Jar-Ferr Yang. Combined 2-D Transform and Quantization Architectures for H.264 Video Coders. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 1802–1805, May 2005.

[93] Tu-Chih Wang, Yu-Wen Huang, Hung-Chi Fang, and Liang-Gee Chen. Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC/H.264. In *International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 800–803, may 2003.

[94] Woong Hwangbo and Chong-Min Kyung. A Multitransform Architecture for H.264/AVC High-Profile Coders. *IEEE Transactions on Multimedia*, 12(3):157–167, April 2010.

[95] R.C. Kordasiewicz and S. Shirani. ASIC and FPGA Implementations of H.264 DCT and Quantization Blocks. In *IEEE International Conference on Image Processing (ICIP)*, volume 3, pages 1020–1023, Sept. 2005.

[96] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro. H.264/AVC Baseline Profile Decoder Complexity Analysis, journal = IEEE Transactions on Circuits and Systems for Video Technology , year = 2003, volume = 13, pages = 704–716, number = 7, month = july, doi = 10.1109/TCSVT.2003.814967, issn = 1051-8215.

[97] M. Shafique, L. Bauer, and J. Henkel. An Optimized Application Architecture of the H.264 Video Encoder for Application Specific Platforms. In *IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, pages 119–124, Oct. 2007.

[98] G. L. Foresti, P. Mahonen, and C. S. Regazzoni. *Multimedia Video-Based Surveillance Systems: Requirements, Issues and Solutions*. Kluwer Academic, 2000.

[99] Joint Video Team. H.264/AVC Reference Software JM 13.2, 2003.

[100] International Technology Roadmap for Semiconductors. `http://www.itrs.net/`, 2010.

[101] Hewlett-Packard Laboratories. VEX Toolchain. `http://www.hpl.hp.com/downloads/vex/`.

List of Publications

International Conferences/Workshops

1. **M. Nadeem,** S. Wong, G. Kuzmanov, A. Shabbir, ***A High-throughput, Area-efficient Hardware Accelerator for Adaptive Deblocking Filter in H.264/AVC***, In Proceedings of IEEE Workshop for Embedded Systems for Real-Time Multimedia (ESTIMedia), pp. 18-27, Grenoble, France, October 2009. citations[1]: 7

2. **M. Nadeem,** S. Wong, G. Kuzmanov, A. Shabbir, M. F. Nadeem, ***Low-power, High-throughput Deblocking Filter for H.264/AVC***, In Proceedings of International Symposium on System-on-Chip (SoC), pp. 93-98, Tampere, Finland, September 2010. citations[1]: 4

3. **M. Nadeem,** S. Wong, G. Kuzmanov, ***An Efficient Realization of Forward Integer Transform in H.264/AVC Intra-frame Encoder,*** In Proceedings of International Symposium on Systems, Architectures, Modeling, and Simulation (SAMOS X), pp. 71-78, Samos, Greece, July 2010. citations[1]: 11

4. **M. Nadeem,** S. Wong, G. Kuzmanov, ***Configurable, Low-power Design for Inverse Integer Transform in H.264/AVC,*** In Proceedings of International Conference on Frontiers of Information Technology (FIT), pp. 32:1-32:5, Islamabad, Pakistan, December 2010. citations[1]: 0

5. **M. Nadeem,** S. Wong, G. Kuzmanov, ***Inverse Integer Transform in H.264/AVC Intra-frame Encoder,*** In Proceedings of International Symposium on Electronic Design, Test and Application (DELTA), pp. 228-233, Queenstown, New Zealand, January 2011. citations[1]: 1

6. **M. Nadeem,** S. Wong, G. Kuzmanov, ***An Efficient Hardware Design for Intra-prediction in H.264/AVC Decoder,*** In Proceedings of International Conference on Electronics, Communications and Photonics (SIECPC), pp. 228-233, Riyadh, Saudi Arabia, April 2011. citations[1]: 2

[1]Citation based on 'Google Scholar' as of 2014/05/06

International Journals

1. **M. Nadeem,** S. Wong, G. Kuzmanov, K.L.M. Bertels, ***Low-complexity, Content-adaptive Design for In-loop Deblocking Filter in H.264/AVC***, to be submitted.

2. **M. Nadeem,** S. Wong, G. Kuzmanov, K.L.M. Bertels, ***On Realization of 2-D Transforms in H.264/AVC Video Codec***, to be submitted.

Patents

1. S. M. Ziauddin, Imran ul-Haq, **M. Nadeem**, M. Shafique, ***Methods and Systems for Providing Low Cost Robust Operational Control for Video Encoders***, Publication Date: September 6th, 2007; Patent Publication No. US-2007-0206674-A1.

Non-related Publications

1. F Anjam, **M. Nadeem**, S. Wong, ***Targeting Code Diversity with Run-time Adjustable Issue-slots in a Chip Multiprocessor***, In Proceedings of the International Conference on Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1358-1363, Grenoble, France, March 2011

2. F Anjam, **M. Nadeem**, S. Wong, ***A VLIW Softcore Processor with Dynamically Adjustable Issue-slots***, In Proceedings of the International Conference on Field Programmable Technology (FPT), pp. 393-398, Beijing, China, December 2010

3. M. F. Nadeem, **M. Nadeem**, S. Wong, ***On Virtualization of Reconfigurable Hardware in Distributed Systems***, In Proceedings of the International Conference on Parallel Processing (ICPP), , pp. 348-356, Pittsburgh, USA, September 2011Z

4. M. F. Nadeem, S. A. Ostadzadeh, **M. Nadeem**, S. Wong, K. L. M. Bertels, ***A Simulation Framework for Reconfigurable Processors in Large-scale Distributed Systems***, In Proceedings of the International Conference on Parallel Processing (ICPP), , pp. 348-356, Taipei City, Taiwan, September 2011

5. M. F. Nadeem, S. A. Ostadzadeh, **M. Nadeem**, M. Ahmadi, S. Wong, ***A Novel Dynamic Task Scheduling Algorithm for Grid Networks with Reconfigurable Processors***, In Proceedings of HiPEAC Workshop on Reconfigurable Computing (WRC), Heraklion, Greece, January 2011

6. M. F. Nadeem, M. Ahmadi, **M. Nadeem**, S. Wong, ***Modeling and Simulation of Reconfigurable Processors in Grid Networks***, In Proceedings of International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 226-231, Cancun, Mexico, December 2010

7. Z. Nawaz, **M. Nadeem**, J. van Someren, K. L. M. Bertels, ***A Parallel FPGA Design of the Smith-Waterman Traceback***, In Proceedings of the International Conference on Field Programmable Technology (FPT), Beijing, China, December 2010

Propositions

accompanying the PhD dissertation

Adaptive, Low-power Architectures for Embedded Multimedia Systems
with focus on H.264/AVC video codec
by Muhammad Nadeem

1. Optimized algorithms with reduced complexity is the key to achieve high performance in terms of throughput, on-chip area, and power consumption in embedded systems. [This thesis]

2. Low-power solutions at the cost of reduced performance or quality are counterproductive. [This thesis]

3. Reconfigurable processors along with application-specific units provide the needed flexibility and performance in energy-constrained embedded systems. [This thesis]

4. Human creativity is nothing but a mapping of a problem-space to a solution-space.

5. Ideas without a thought to implementation are of little use.

6. For a nation to make progress, moral values and positive attitudes are far more important than anything else.

7. Distress is the difference between two wills: ours and that of God's. [Anonymous]

8. The number of revisions of a PhD draft thesis are directly proportional to the number of visits to the supervisor.

9. Dreams are a pre-requisite for a PhD. If you do, then all you need is a supervisor to wake you up and ask you to make it a reality.

10. No matter what religion you practice, you'll find a worship place of your preference in Netherlands. This is not only because the Dutch people are tolerant but mostly because they have a different opinion on everything.

These propositions are regarded as opposable and defendable, and have been approved as such by the promoter, Prof.dr. K.L.M. Bertels.

Stellingen

behorende bij het proefschrift

Adaptive, Low-power Architectures for Embedded Multimedia Systems
with focus on H.264/AVC video codec
door Muhammad Nadeem

1. Geoptimaliseerde algoritmen door gereduceerde complexiteit is de sleutel tot het realiseren van high performance in termen van throughput, on-chip oppervlakte, en het energieverbruik in embedded systemen [Deze dissertatie]
2. Low-power oplossingen die ten koste gaan van prestatie of kwaliteit zijn contraproductief. [Deze dissertatie]
3. Herconfigureerbare processoren leveren samen met applicatie-specifieke units de benodigde flexibiliteit en prestaties in energie-begrensde embedded systemen. [Deze dissertatie]
4. Menselijke creativiteit is niets anders dan het afbeelden van een probleemruimte naar een oplossingsruimte.
5. Ideeën zonder gedachte te hebben aan implementatie zijn van weinig nut.
6. Om als volk vooruitgang te boeken zijn morele waarden en positieve opvattingen veel belangrijker dan wat dan ook.
7. Benardheid is het verschil tussen twee wensen: de onze en die van God. [Anoniem]
8. Het aantal revisies van een doctoraal concept proefschrift zijn recht evenredig met het aantal bezoeken aan de supervisor.
9. Dromen zijn vereist voor een PhD. Als je dat doet, dan is alles wat je nodig hebt een begeleider om je te ontwaken en je te vragen om het een realiteit te maken.
10. Welke religie je ook beoefent, je zult een bidplaats vinden van jouw voorkeur in Nederland. Dit is niet alleen omdat de Nederlandse mensen tolerant zijn, maar vooral omdat ze over alles van mening verschillen.

Deze stellingen worden opponeerbaar en verdedigbaar geacht en zijn als zodanig goedgekeurd door de promotor, Prof.dr. K.L.M. Bertels.

Curriculum Vitae

Muhammad Nadeem was born on October 1, 1970 in Multan, Pakistan. He received his B.Sc. and M.Sc. degrees from Bahauddin Zakariya University (BZU), Multan, Pakistan, Quaid-i-Azam University (QAU), Islamabad, Pakistan in 1991 and 1994 respectively. Later in 1995, after completing his post graduate diploma in Computer Systems Software/Hardware, from Computer Training Center (CTC) Islamabad, he worked as Scientific Officer in a research and development organization in Pakistan between 1995-1998. From 1998-2013, he worked for several well-known international companies in USA and Europe (incl. Lucent Technologies, NXP Software, Mapper Lithography, Cochlear etc).

Mr. Nadeem has accrued 15+ years' design, development and research experience in real-time embedded systems in different domains, like audio/video compression technologies, tele-communications, and low-power medical implants (covering both hardware & software perspectives) in leading industrial and research/educational organizations. He is always interested in, and posses valuable experience in different phases of the product development, from concept to its realization and has contributed in number of successful products on the market. His main research interests includes signal processing applications/algorithms, architectures, and design methodologies for embedded systems based on Digital Signal Processors (DSP), multi/many-core extensible and reconfigurable processors with a focus on high-performance, low-power, and reliability.

In 2006, he joined Computer Engineering Lab, in EEMCS faculty at Delft University of Technology for pursuing his PhD. He worked on adaptive, low-power architectures for embedded multimedia systems; with focus on H.264/AVC video codec, under the supervision of Prof. Stamatis Vassiliadis and Prof. Koen Bertels. He was co-supervised by Dr. J.S.S.M. Wong. The research conducted by him is presented in this dissertation.