UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CLÁUDIO MACHADO DINIZ

Dedicated and Reconfigurable Hardware Accelerators for High Efficiency Video Coding Standard

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: Prof. Dr. Sergio Bampi

Porto Alegre
2015

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Diniz, Cláudio Machado
Dedicated and Reconfigurable Hardware Accelerators for High Efficiency Video Coding Standard / Cláudio Machado Diniz. – 2015.
141 f.

Orientador: Sergio Bampi.

Tese (Doutorado) – Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2015.

1. HEVC. 2. Hardware Accelerator 3. Video Coding Architecture. 4. Reconfigurable Architectures. I. Bampi, Sergio, orient. II. Título.

This thesis is dedicated to my wife, Cilene,
and to my mother, Eliane.

Dedico esta tese à minha noiva, Cilene,
e à minha mãe, Eliane.

Ich sage euch: man muss noch Chaos in sich haben,

um einen tanzenden Stern gebären zu können.

Ich sage euch: ihr habt noch Chaos in euch.

Friedrich Nietzsche

AGRADECIMENTOS

O melhor deste longo e intenso período de estudos de doutorado foi conviver com uma quantidade grande, e sempre crescente, de pessoas incríveis. Gostaria de agradecer, de coração, a essas pessoas especiais que me ajudaram, das mais diversas formas, no desenvolvimento deste trabalho, e que, de certa forma, também possam compartilhar comigo a alegria de ter conquistado mais esta etapa.

Quero agradecer de coração à minha noiva, Cilene, por seu apoio e amor incondicional. Por sempre me dar força e incentivo em todas minhas ações e decisões, mesmo que elas significassem a nossa distância física por longos períodos (como foi durante o estágio na Alemanha). Pela compreensão quando eu estava distante por causa do trabalho. Pelo seu sorriso e aquela tentativa de me deixar mais feliz em momentos difíceis. Por sempre compartilhar comigo também dos momentos felizes. Faltam-me até palavras para tudo que tenho a agradecê-la. Eu te amo, Cilene.

Quero agradecer de coração à minha mãe, Eliane, que me transmitiu todos os valores e a educação que carrego sempre comigo. Diante de todas as dificuldades que passamos, ela esteve sempre forte e determinada em seguir em frente e contribuir para minha formação. Agradeço pelas palavras de apoio, que me ajudaram a recuperar a autoconfiança. Agradeço a ela por ser uma mãe maravilhosa e uma profissional na qual eu tento me espelhar, um exemplo de responsabilidade e honestidade.

Agradeço ao meu irmão, Fernando, pela convivência durante os anos em que moramos juntos e pela amizade de sempre. Quero agradecer também aos meus sogros, Carmen e Cilon, pela amizade e por toda ajuda ao longo destes anos.

Agradeço ao meu orientador, professor Sergio Bampi, por ter primeiramente confiado no meu potencial e me aceitado no seu grupo de pesquisa. Agradeço por toda sua orientação, ensinamentos, e ajuda, sejam elas técnicas ou pessoais. Agradeço pelas oportunidades que me foram proporcionadas, as quais foram muito enriquecedoras para minha formação como pesquisador e como pessoa.

Agradeço aos professores Altamiro Susin e Luciano Agostini, por terem escrito, em duas ocasiões, cartas de recomendação nas minhas duas tentativas em ingressar no doutorado em Computação da Universidade Federal do Rio Grande do Sul (UFRGS).

Agradeço a todos os colegas e amigos do meu grupo de pesquisa da UFRGS, conhecido informalmente como “lab 215”, pela amizade e por todas as discussões

técnicas e não técnicas. Agradeço em especial aos colegas Bruno Zatt, Guilherme Corrêa, Eduarda Monteiro, Daniel Palomino, Felipe Sampaio, Mateus Grellert, Leonardo Soares, André Rosa, Bruno Vizzotto, Roger Porto, Fábio Ramos, Vagner Rosa, Débora Matos, Leandro Max, Kleber Stangherlin e Fábio Walter. Agradeço também a todos os alunos de graduação que passaram no grupo de pesquisa como bolsistas de iniciação científica ou para desenvolver trabalho de conclusão de curso. Agradeço em especial ao Felipe Dalcin e Filipe Posteral, cujos trabalhos de conclusão eu tive a oportunidade de co-orientar, e que contribuíram para o desenvolvimento desta tese.

Agradeço aos amigos e colegas que conheci (ou reencontrei) no curto período em que trabalhei na empresa CEITEC S.A em 2012. Agradeço em especial aos colegas do meu time, Fábio Ramos, Marcos Hervé, Frederico Moller, Daniel Ferrão, Marcelo Moraes, Janaína Costa, e meu chefe, Murugappan Ramaswami, pelos ensinamentos sobre projeto de circuitos digitais.

Durante o estágio na Alemanha, em 2013, pude conviver também com pessoas incríveis. Agradeço ao professor Jörg Henkel, por ter me recebido no Chair for Embedded Systems do Karlsruhe Institute of Technology (KIT), em Karlsruhe, e por ter me proporcionado um excelente ambiente de trabalho para meu estágio de doutorado-sanduíche em 2013. Aprendi muito sobre como pesquisar durante o estágio no CES/KIT. Agradeço, em especial, ao Dr. Muhammad Shafique, que me orientou e discutiu comigo os trabalhos de pesquisa que realizei durante o estágio e mesmo os demais trabalhos iniciados depois do estágio. Não é por acaso que praticamente todos os artigos que foram produzidos nesta tese tem o prof. Henkel e o Dr. Shafique como co-autores.

Agradeço a todo esforço realizado pelo colega e amigo Bruno Zatt que, juntamente com o Dr. Shafique, deu início ao processo de cooperação científica entre nosso grupo da UFRGS e o grupo do CES/KIT. Este processo de cooperação iniciado resultou em uma cooperação bilateral formal, financiada pela CAPES, que me apoiou com uma bolsa de estudos de um ano na Alemanha.

Agradeço aos amigos José Azambuja e Georg Hasenpflug, pela excelente convivência diária no ano em que morei em Karlsruhe. Especialmente o José me ajudou muito na mudança para esta nova cidade e país, conseguindo um quarto no apartamento em que morava, me orientando nos primeiros passos no novo país, entre outras coisas.

Gostaria de agradecer às visitas da minha noiva Cilene, da minha mãe Eliane, da minha tia Hilda, e de meu amigo Felipe Sampaio. Elas tornaram meus dias mais agradáveis, diante da saudade que eu sentia dos familiares e amigos do Brasil naquele momento. Agradeço ainda aos amigos Oliver Longhi, Mateus Grellert, José Amendola, Daniel Palomino, Arthur Veiga, Luiza Biasoto, Gabriel Marchesan e família, Philip Porter e Abelardo Gonzalez, pela amizade e companhia em Karlsruhe. Agradeço também aos meus colegas do CES/KIT, especialmente Muhammad Usman Khan, Farzad Samie Ghahfarokhi, Fazal Hameed e Daniel Palomino, pela amizade e conversas diárias.

Agradeço aos professores da Universidade Católica de Pelotas (UCPel), Eduardo Costa, Sergio Almeida, Leandro Zafalon, Wemerson Parreira, Adenauer Yamin e Monica Matzenauer, que pude conhecer ou reencontrar em 2014, quando comecei a trabalhar na UCPel. Especialmente o professor Eduardo Costa me deu grande apoio para que eu pudesse focar na escrita de artigos e no término desta tese. Agradeço também ao professor Mateus Beck Fonseca, da Universidade Federal de Pelotas (UFPel), com quem tive a oportunidade de começar um trabalho ainda em 2014 que resultou em uma contribuição para esta tese, e que me auxiliou na parte experimental do projeto de circuitos digitais.

Agradeço a todos os professores e funcionário do Programa de Pós-Graduação em Computação e do Instituto de Informática da UFRGS, que propiciaram um excelente ambiente para pesquisa.

Agradeço aos órgãos de fomento, CNPq e CAPES, por apoiarem meus estudos através da concessão de bolsas de estudo no país e no exterior, respectivamente.

Por fim, agradeço a todos familiares e amigos que não foram citados nominalmente.

A todos, meu muito obrigado!

ACKNOWLEDGEMENTS

The best thing of this long and intense period of Ph.D. studies was to meet a large (and always growing) set of incredible people. I would like to thank those special people that helped me, in many different ways, in the development of this work. I hope they can share with me the joy of achieving this step.

I would like to thank my wife, Cilene, for her support and unconditional love. For always support and encourage all my actions and decisions, even if they mean our long distance for long periods (which was the case during the internship in Germany). For her understanding when I was distant because of work. For her smile and that effort to make me happier in difficult moments. For always share with me also the moments of joy. I can't thank you enough for everything you have done. I love you, Cilene.

I would like to thank my mother, Eliane, who transmitted all the human values and education that I carry with me. In the face of all the difficulties we went through, she was always strong and determined to move forward and contribute to my education. I thank for all the words of support, which helped me to recover my self-assurance. I thank her for being a wonderful mother and also a professional that I try to look up to, an example of responsibility and honesty.

I thank my brother, Fernando, for being my roommate during some years in Porto Alegre and for his friendship. I would like to thank also my mother-in-law and father-in-law, Carmen e Cilon, for their friendship and all the help during those years.

I thank my advisor, professor Sergio Bampi, for trusting in my potential and accepting me in his research group. I thank for all the guidance, teaching, and the technical and personal help. I thank for all the opportunities he provided me. They were very enriching for me as researcher and person.

I thank to the professors Altamiro Susin and Luciano Agostini, for writing, in two occasions, recommendation letters in my two attempts to join the Ph.D. program in Computer Science at the Federal University of Rio Grande do Sul (UFRGS).

I thank all my colleagues and friends of my research group in UFRGS, informally known as "lab 215", for the friendship and for all the technical and non-technical discussions. I thank in particular the colleagues Bruno Zatt, Guilherme Corrêa, Eduarda Monteiro, Daniel Palomino, Felipe Sampaio, Mateus Grellert, Leonardo Soares, André Rosa, Bruno Vizzotto, Roger Porto, Fábio Ramos, Vagner Rosa, Débora

Matos, Leandro Max, Kleber Stangherlin and Fábio Walter. I also thank all the undergraduate students that worked in the research group, in special Felipe Dalcin and Felipe Posteral, whose final undergraduate works I had the opportunity to co-advise, and which contributed to the development of this thesis.

I thank all the friends and colleagues that I met in the short period I worked in CEITEC S.A company in 2012. I thank in particular the colleagues of my team, Fábio Ramos, Marcos Hervé, Frederico Moller, Daniel Ferrão, Marcelo Moraes, Janaína Costa, and my boss, Murugappan Ramaswami, for teaching me valuable things about digital circuit design.

During my internship in Germany, in 2013, I also had the opportunity to meet incredible people. I thank to the professor Jörg Henkel, for receiving me in the Chair for Embedded Systems of the Karlsruhe Institute of Technology (KIT), in Karlsruhe, and for providing me an excellent work office for my internship. I learned a lot how to research during my internship in CES/KIT. I would like to thank, in particular, Dr. Muhammad Shafique, who guided me and discussed with the research projects that I realized during the internship and even the works that began after the internship. It is no coincidence that prof. Henkel and Dr. Shafique are co-authors in most papers that have been produced in this thesis.

I thank all the effort realized by my colleague and friend Bruno Zatt that, along with Dr. Shafique, began the scientific cooperation process between our research group at UFRGS and the CES/KIT research group. This cooperation process resulted in a formal bilateral scientific cooperation agreement, funded by CAPES, which supported me with an one-year research scholarship in Germany.

I thank to my friends José Azambuja and Georg Hasenpflug, for the daily friendship during the year that we were roommates in Karlsruhe. In particular, José helped me a lot in my change for this new city and country, getting a room for me in the apartment he lived that time, helping me in the first steps in Germany, and other things. I would like to thank the visits of my wife Cilene, my mother Eliane, my aunt Hilda, and my friend Felipe Sampaio. I missed my family and friends that time, and those visits make my days more enjoyable. I thank my friends Oliver Longhi, Mateus Grellert, José Amendola, Daniel Palomino, Arthur Veiga, Luiza Biasoto, Gabriel Marchesan e família, Philip Porter and Abelardo Gonzalez, for the friendship in Karlsruhe. I also thank my colleagues from CES/KIT, in particular Muhammad Usman

Khan, Farzad Samie Ghahfarokhi, Fazal Hameed and Daniel Palomino, for the friendship and daily discussions.

I thank the professors from Catholic University of Pelotas (UCPel), Eduardo Costa, Sergio Almeida, Leandro Zafalon, Wemerson Parreira, Adenauer Yamin and Monica Matzenauer, which I met in 2014 when I started working in this university. In particular, professor Eduardo Costa gave me a lot of support so I could focus in writing papers and finishing my thesis. I also thank professor Mateus Beck Fonseca, from the Federal University of Pelotas (UFPel), with whom I had the opportunity to start a work in 2014 that resulted in a contribution of this thesis. He also helped me in the experimental part of digital circuit design.

I thank all the professors and staff of Graduate Program in Computer Science and the Informatics Institute of UFRGS, which provided an excellent environment for research.

I thank the funding agencies, CNPq and CAPES, for supporting my studies through research scholarships in Brazil and in Germany, respectively.

Finally, I thank all my relatives and friends that were not nominally mentioned.

To all, thank you very much!

ABSTRACT

The demand for ultra-high resolution video (beyond 1920x1080 pixels) led to the need of developing new and more efficient video coding standards to provide high compression efficiency. The High Efficiency Video Coding (HEVC) standard, published in 2013, reaches double compression efficiency (or 50% reduction in size of coded video) compared to the most efficient video coding standard at that time, and most used in the market, the H.264/AVC (Advanced Video Coding) standard. HEVC reaches this result at the cost of high computational effort of the tools included in the encoder and decoder. The increased computational effort of HEVC standard and the power limitations of current silicon fabrication technologies makes it essential to develop hardware accelerators for compute-intensive computational kernels of HEVC application. Hardware accelerators provide higher performance and energy efficiency than general purpose processors for specific applications. An HEVC application analysis conducted in this work identified the most compute-intensive kernels of HEVC, namely the Fractional-pixel Interpolation Filter, the Deblocking Filter and the Sum of Absolute Differences calculation. A run-time analysis on Interpolation Filter indicates a great potential of power/energy saving by adapting the hardware accelerator to the varying workload. This thesis introduces new contributions in the field of dedicated and reconfigurable hardware accelerators for HEVC standard. Dedicated hardware accelerators for the Fractional Pixel Interpolation Filter, the Deblocking Filter and the Sum of Absolute Differences calculation are herein proposed, designed and evaluated. The interpolation filter hardware architecture achieves throughput similar to the state of the art, while reducing hardware area by 50%. Our deblocking filter hardware architecture also achieves similar throughput compared to state of the art with a 5X to 6X reduction in gate count and 3X reduction in power dissipation. The thesis also does a new comparative analysis of Sum of Absolute Differences processing elements, in which various architecture design alternatives with different area, performance and power results were introduced. A novel reconfigurable interpolation filter hardware architecture for HEVC standard was developed, and it provides 57% design-time area reduction and run-time power/energy adaptation in a picture-by-picture basis, compared to the state-of-the-art. Additionally a run-time accelerator binding scheme is proposed for tile-based mixed-grained reconfigurable architectures, which

reduces the communication overhead, compared to first-fit strategy with datapath reusing scheme, by up to 44% (23% on average) for different number of tiles and internal tile organizations. This run-time accelerator binding scheme is aware of the underlying architecture to bind datapaths in an efficient way, to avoid and minimize inter-tile communications. The new dedicated and reconfigurable hardware accelerators and techniques proposed in this thesis enable next-generation video coding standard implementations beyond HEVC with improved area, performance, and power efficiency.

Keywords: HEVC, Hardware Accelerator, Video Coding Architecture, Reconfigurable Architectures.

Aceleradores Dedicados e Reconfiguráveis para o Padrão High Efficiency Video Coding (HEVC)

RESUMO

A demanda por vídeos de resolução ultra-alta (além de 1920x1080 pontos) levou à necessidade de desenvolvimento de padrões de codificação de vídeo novos e mais eficientes para prover alta eficiência de compressão. O novo padrão High Efficiency Video Coding (HEVC), publicado em 2013, atinge o dobro da eficiência de compressão (ou 50% de redução no tamanho do vídeo codificado) comparado com o padrão mais eficiente até então, e mais utilizado no mercado, o padrão H.264/AVC (Advanced Video Coding). O HEVC atinge este resultado ao custo de uma elevação da complexidade computacional das ferramentas inseridas no codificador e decodificador. O aumento do esforço computacional do padrão HEVC e as limitações de potência das tecnologias de fabricação em silício atuais tornam essencial o desenvolvimento de aceleradores de hardware para partes importantes da aplicação do HEVC. Aceleradores de hardware fornecem maior desempenho e eficiência energética para aplicações específicas que os processadores de propósito geral. Uma análise da aplicação do HEVC realizada neste trabalho identificou as partes mais importantes do HEVC do ponto de vista do esforço computacional, a saber, o Filtro de Interpolação de Ponto Fracionário, o Filtro de Deblocagem e o cálculo da Soma das Diferenças Absolutas. Uma análise de tempo de execução do Filtro de Interpolação indica um grande potencial de economia de potência/energia pela adaptação do acelerador de hardware à carga de trabalho variável. Esta tese introduz novas contribuições no tema de aceleradores dedicados e reconfiguráveis para o padrão HEVC. Aceleradores de hardware dedicados para o Filtro de Interpolação de Pixel Fracionário, para o Filtro de Deblocagem, e para o cálculo da Soma das Diferenças Absolutas, são propostos, projetados e avaliados nesta tese. A arquitetura de hardware proposta para o filtro de interpolação atinge taxa de processamento similar ao estado da arte, enquanto reduz a área do hardware para este bloco em 50%. A arquitetura de hardware proposta para o filtro de deblocagem também atinge taxa de processamento similar ao estado da arte com uma redução de 5X a 6X na contagem de gates e uma redução de 3X na dissipação de potência. A nova análise comparativa proposta para os elementos de processamento do cálculo da Soma das Diferenças Absolutas introduz diversas alternativas de projeto de arquitetura com

diferentes resultados de área, desempenho e potência. A nova arquitetura reconfigurável para o filtro de interpolação do padrão HEVC fornece 57% de redução de área em tempo de projeto e adaptação da potência/energia em tempo-real a cada imagem processada, o que ainda não é suportado pelas arquiteturas do estado da arte para o filtro de interpolação. Adicionalmente, a tese propõe um novo esquema de alocação de aceleradores em tempo-real para arquiteturas reconfiguráveis baseadas em tiles de processamento e de grão-misto, o que reduz em 44% (23% em média) o “overhead” de comunicação comparado com uma estratégia first-fit com reuso de datapaths, para números diferentes de tiles e organizações internas de tile. Este esquema de alocação leva em conta a arquitetura interna para alocar aceleradores de uma maneira mais eficiente, evitando e minimizando a comunicação entre tiles. Os aceleradores e técnicas dedicadas e reconfiguráveis propostos nesta tese proporcionam implementações de codificadores de vídeo de nova geração, além do HEVC, com melhor área, desempenho e eficiência em potência.

Palavras-Chave: HEVC, Acelerador de Hardware, Arquitetura para Codificação de Vídeo, Arquiteturas Reconfiguráveis.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABREVIATIONS AND ACRONYMS

AGU	Address Generation Unit
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
AVC	Advanced Video Coding
B	Bi-predictive
BD-PSNR	Bjontegaard Delta Peak Signal-to-Noise Ratio
BD-Rate	Bjontegaard Delta Rate
BRAM	Block Random Access Memory
CABAC	Context-Adaptive Binary Arithmetic Coding
CAVLC	Context-Adaptive Variable Length Coding
Cb	Chrominance Blue
CCD	Charge Coupled Devices
CG	Coarse-grained
CLB	Configurable Logic Block
CMOS	Complementary metal-oxide-semiconductor
CODEC	COder/DECoder
Cr	Chrominance Red
CTU	Coding Tree Unit
CU	Coding Unit
DCT	Discrete Cosine Transform
DF	Deblocking Filter
DC	Direct Current
MOS	Metal Oxide Semiconductor
DP	Datapath
DST	Discrete Sine Transform
DVD	Digital Versatile Disk
EP	Estimated Performance
EXOR	Exclusive OR
FG	Fine-grained
FME	Fractional Motion Estimation
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine

GB	Gigabytes
GOP	Group of Pictures
GPB	Generalized P and B
HD	High Definition
HEVC	High Efficiency Video Coding
HM	HEVC Test Model
ID	Interpolation Datapath
IME	Integer Motion Estimation
IP	Intra-Period
IQ	Inverse Quantization
ISO	International Organization for Standardization
IT	Inverse Transforms
ITU	International Telecommunication Union
JCT-VC	Joint Collaborative Team on Video Coding
JVT	Joint Video Team
LD	Low Delay
LUT	Look-Up Table
MC	Motion Compensation
MD	Manhattan Distance
ME	Motion Estimation
MNFC	Monitored Number of Filter Calls
MOS	Mean Opinion Score
MPEG	Motion Picture Experts Group
MSE	Mean-Squared Error
MV	Motion Vector
NRE	Non-Recurring Engineering
P	Predictive
PDP	Power Delay Product
PE	Processing Element
PNFC	Predicted Number of Filter Calls
POC	Picture Order Count
PRR	Partial Run-Time Reconfiguration
PSNR	Peak Signal-to-Noise Ratio
PU	Prediction Unit
Q	Quantization

QFHD	Quad Full High Definition
QP	Quantization Parameter
QVGA	Quarter Video Graphics Array
RA	Random Access
RAM	Random Access Memory
RCA	Ripple-Carry Adders
RGB	Red, Green, and Blue
RTL	Register Transfer Level
SAD	Sum of Absolute Differences
SAO	Sample Adaptive Offset
SATD	Sum of Absolute Transformed Differences
SD	Standard Definition
SoC	Systems-on-Chip
T	Transforms
TU	Transform Unit
VCEG	Video Coding Experts Group
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
WPP	Wavefront Parallel Processing
WQVGA	Wide Quarter Video Graphics Array
WVGA	Wide Video Graphics Array
Y	Luminance
YCbCr	Luminance, Chrominance Blue, Chrominance Red

TABLE OF CONTENTS

1 INTRODUCTION

Nowadays, there are many devices in the market capable of digital video recording and displaying, such as digital (smart) televisions, desktop and laptop computers, tablets, smartphones, videogame consoles, camcorders, security cameras, etc. These devices enable a variety of digital video applications, such as video streaming, broadcast digital television, videoconferencing, digital cinema, video surveillance, etc. Two on-demand digital video streaming services over the Internet, namely YouTube and Netflix, became increasingly popular in the last years. YouTube is the largest video repository and video broadcast service in the Internet, with 80 hours of video uploaded per minute by the users and millions of views per day (KOKARAM, 2013). Netflix is subscription-based streaming video service that delivers movies and TV series. Netflix achieved the mark of 50 million subscribers in the second quarter of 2014 (FORBES, 2014). It is predicted that video traffic over the Internet will be 79% of all consumer Internet traffic in 2018 (CISCO, 2014).

To deal with video storing and transmitting over the Internet (and other communication networks), video compression is essential. Here is an example of why video compression is important: a raw (uncompressed) video lasting 10 minute with 720x480 pixel resolution (Standard Definition - SD) represented with 24 bits per pixel (8 bit for each color channel, using three color channels) and with 30 frames per second (fps) require 19 Gigabytes (GB) to be stored or transmitted over the Internet. The same 10-minute raw video with 1920x1080 pixels resolution (Full-HD resolution) requires 112 GB. The same video in the new Sony 4K video resolution format (4096x2160 pixels), used in the 2014 FIFA World Cup, requires 477 GB. It is not viable to deal with such amount of data of raw video sequences using the recent storing and communication technologies.

Video coding is the process of compressing and decompressing digital video. In other words, video coding is the process of converting digital video into a format suitable for transmission or storage. The number of bits to represent encoded video is reduced compared to raw video. Video coding is based in a complementary pair of systems, an encoder (compressor) and a decoder (decompressor). Video encoder converts raw video into a compressed form, prior to storing or transmission. This process is also known as video encoding. Video decoder converts the compressed video back to the original (or very similar to the original) video representation. This process is also known as video decoding. The encoder/decoder pair is often described as a CODEC (enCOder/DECoder). Video compression is achieved by removing redundancy, i.e. information that is not necessary for video representation. Video compression may also introduce subjective redundancy, i.e. information that can be removed without significantly affecting viewer's perception of video quality. If the decoded video is identical than original raw video, the encoding process is lossless. In lossy compression, subjective redundancy is also employed, resulting in difference between raw video and decoded video. Lossy compression is applied to achieve higher compression. The higher compression comes with a decrease in video quality of decoded video compared with raw video (RICHARDSON, 2010).

Video coding standards are developed to encode (compress) video. Most video coding standards employ lossy compression to attain high video compression efficiency. When developing a video coding standard, the goal is to compress video with minimal video quality loss under a certain compressed video size (or to achieve minimum compressed video size under a given target video quality). Video coding standards evolved in the last two decades, primarily driven by new video applications and the increase in video resolution. The advances of recent video coding standards in order to provide increasingly video compression result in a huge computational effort. Electronic devices capable of video processing are demanded to provide increased performance at each video coding standard generation, to encode and decode high resolution videos in real time. In this context, section 1.1 presents motivation and problem definition which are driving this thesis.

1.1 Motivation and Problem Definition

The recent demand for ultra-high resolution videos (beyond 1920x1080 pixels) drives the development of new and more efficient video coding standards to provide high compression efficiency. The most efficient new coding standard that arises is the High Efficiency Video Coding (HEVC) standard, developed by the Joint Collaborative Team on Video Coding (JCT-VC), formed by experts of Video Coding Experts Group (VCEG) of International Telecommunication Union (ITU), and Motion Picture Experts Group (MPEG) from the International Standardization Union (ISO). HEVC was published in April 2013 as ITU-T H.265 recommendation (ITU-T, 2013).

HEVC reaches the double compression efficiency (or 50% bit rate reduction) compared to the most efficient video coding standard at that time, and most used in the market, the H.264/AVC (Advanced Video Coding) standard (ITU-T, 2011). The double compression efficiency of HEVC over H.264/AVC is achieved for a similar video quality, since both standards provide lossy compression. HEVC achieves such compression efficiency by employing larger block sizes (to deal with increased resolutions), sophisticated block partitioning, and new advanced coding tools (SULLIVAN, 2012).

The higher compression efficiency of HEVC comes with a significant increase in the computational effort of the HEVC encoder that ranges from 1.2x to 3.2x of the H.264/AVC encoding complexity (VANNE, 2012). It requires further performance improvement of video-capable devices to deal with the increased encoding complexity still being able to encode high video resolutions in real time. A substantial research effort, especially in HEVC encoder, is forecasted to reach this goal (BOSSEN, 2012).

Performance improvement was achieved in the past with the advances of silicon fabrication technology, which enable higher operation frequencies through smaller and faster transistors. Recently, advances in silicon fabrication still enable smaller and faster transistors at every new Complementary metal-oxide-semiconductor (CMOS) technology node. Chips continue to integrate more transistors into the same area, following the Moore's law for density (MOORE, 1965). Recent CMOS technologies are able to integrated more and more processing cores in the same chip, the so-called multi-core and many-core processors.

However, in new decananometer technologies, performance increase is limited by thermal design power, since transistor power density is now increasing at each CMOS technology node (ESMAEILZADEH, 2011). To ensure that chips remain below the

thermal design power, not all transistors of the chip can switch at full speed all the time, resulting in the so-called utilization wall (VENKATESH, 2010). The portion of the chip that is most of the time underclocked or powered-off is referred in general as “dark silicon” (ESMAEILZADEH, 2011) (TAYLOR, 2013). Recent work (TAYLOR, 2013) foresees that the percentage of dark silicon will increase at each technology node and will be around 94% in the early 2020’s. Dark silicon will limit core and frequency scaling for performance improvement.

To cope with the increased performance required by the new HEVC standard and to keep chips below the thermal design power, future processors will integrate many on-chip hardware accelerators for specific computational kernels, i.e. the so-called accelerator-rich architectures (IYER, 2012) (CONG, 2014). Specialized hardware accelerators are 500X more energy efficient than general purpose processors doing the same job (HAMMED, 2011). As the computational kernels are not executed simultaneously all the time, the accelerators can be powered off when not in use. Hence, specialized hardware accelerators for important computational kernels are an efficient way to “fill” the dark region of chips.

While dedicated hardware accelerators provide high performance and energy efficiency for real-time video encoding and decoding, they have some drawbacks. They are fixed in design time and cannot change the hardware in the field, after silicon fabrication. They also incur in high costs for design and silicon fabrication. Reconfigurable hardware provides a platform solution with low design costs, faster time-to-market, and flexibility of quick upgrades through dynamic reconfigurations (TUAN, 2006). Designs based on Field-programmable Gate Array (FPGA) combine the performance efficiency of dedicated accelerators due to their capability to exploit high degree of parallelism along with a high degree of flexibility due to their programmability and hardware reconfigurability (SHAFIQUE, 2009)(COMPTON, 2002). The drawback of FPGA designs is the higher power compared with dedicated accelerators.

The thesis author foresees that both dedicated and reconfigurable accelerators will be used in future accelerator-rich processor architectures. This thesis provides different contributions for both dedicated and reconfigurable acceleration, as discussed in section 1.2., which can be applied to the current HEVC and also to future generation video encoders, as long as they are based on CUs comprised of blocks of video pixels.

1.2 Thesis Contribution

The goal of this thesis is the research on novel dedicated and reconfigurable hardware accelerators for important computational kernels of the new HEVC standard. Analysis of the HEVC application presented in Chapter 3 verifies that the most important coding tools in terms of computational effort are the Fractional-pixel Interpolation Filter, the Deblocking Filter (DF) and the Sum of Absolute Differences (SAD).

In the domain of dedicated accelerators dealt with in Chapter 4, this thesis introduces the following novel contributions:

- A high-throughput hardware architecture for HEVC Fractional-pixel Interpolation Filter (Section 4.2) employing 12-pixel parallel filter acceleration engines for luminance and chrominance with multiplierless configurable interpolation datapaths and a scheduling scheme to manage the

operation of these interpolation datapaths depending upon the execution scenario.

- A hardware architecture for HEVC Fractional-pixel Interpolation Filter using Adder Compressors (Section 4.3) to improve area, performance, and power dissipation.
- A high-throughput hardware architecture for HEVC Deblocking Filter (Section 4.4) employing hardware reuse to accelerate filtering decision units at low area cost.
- A comparative analysis and comparison of architectural alternatives for SAD hardware architecture processing element (Section 4.5) in terms of hardware area, throughput, and power dissipation.

Regarding reconfigurable video coding accelerators, this thesis additionally introduces the following novel contributions:

- A reconfigurable hardware architecture for HEVC Fractional-pixel Interpolation Filter, developed in Chapter 5. The architecture incorporates a prediction scheme to estimate the number of interpolation filter calls on a picture-by-picture basis, using the knowledge of the coding structure. A set of different implementation versions for interpolation filter hardware accelerator engines is developed to allow an area versus performance tradeoff. An implementation version selection scheme is proposed. It selects an implementation version of interpolation filter accelerator for each picture, depending upon the predicted number of interpolation filter calls provided by the prediction scheme. A scheduling scheme is introduced to determine the order of processing and configure filter types. It facilitates the reuse of input samples and prevents redundant fetching.
- A run-time accelerator binding scheme for tile-based mixed-grained reconfigurable architectures is proposed and analyzed in Chapter 6. The scheme employs datapath reuse and inter-tile communication cost estimation to perform a communication-minimizing binding for datapaths of custom instructions at run-time.

1.3 Thesis Outline

This text is organized as follows. Chapter 2 provides a background of video coding (focusing on the new HEVC video coding standard), reconfigurable computing, power dissipation and it discusses related work on these topics. Chapter 3 presents an analysis of HEVC application based on software profiling and observation of function calls at run time. Chapter 4 presents our novel dedicated hardware accelerators for HEVC. Chapter 5 presents a new reconfigurable hardware architecture for HEVC Fractional-pixel Interpolation Filter. Chapter 6 introduces the novel run-time accelerator binding scheme for tile-based mixed-grained reconfigurable processors. Chapter 7 presents the conclusions of this thesis and describes future work.

2 BACKGROUND AND RELATED WORK

This chapter gives a background on the concepts required to understand the novel contributions of this work. It revises the concepts of digital video capture, representation, quality and coding (sections 2.1 and 2.2) with an overview of the new HEVC standard (section 2.3). Concepts of reconfigurable computing and power dissipation in CMOS are also revised in sections 2.4 and 2.5. Related work about hardware design and architectures for HEVC and accelerator binding schemes are presented in section 2.6.

2.1 Digital Video Capture, Representation, and Video Quality

2.1.1 Digital Video Capture

Digital video is a discrete approximation (in time and space) of a natural scene. It is represented by a sequence of pictures (a rectangular matrix of pixels) captured at regular time interval. Each picture is 2D projection of a natural scene captured by an analog semiconductor sensor, formed by an array of Charge Coupled Devices (CCD) (RICHARDSON, 2010). Each CCD captures one pixel. For color images, there are three matrixes of CCDs, one to capture each color. Each color component of a pixel is called sample. Each sample is represented by a number of bits, e.g. 8 bits, that define the intensity level of the specific color.

The size of CCD array (in horizontal and vertical pixels) defines the spatial sampling of video, which is called resolution. There are some defined resolution formats, e.g. Standard Definition (SD) with 720x480 pixels, Full-HD (High Definition) with 1920x1080 pixels, 4Kx2k with 4096x2160 pixels, etc.

The time interval each picture is captured define the temporal sampling of video. It is called picture rate or frame rate, commonly defined as frames per second (fps). With a sufficient high capture rate, it is possible to give the observer the feeling of motion. Common picture rates are 24 fps, 30 fps, 50 fps and 60 fps, etc. The higher is the picture rate, more smooth is the feeling of motion to the observer (RICHARDSON, 2010).

2.1.2 Color Spaces and Color Sub-sampling

The pixel in a color picture is represented by three color components, following a color space. A common color space is Red, Green, and Blue (RGB), which uses these three primary colors captured by the human visual system to form the pixel. Video coding standards uses Luminance, Chrominance Red and Chrominance Blue (YCbCr) color space, instead. The main advantage of YCbCr color space is because the human visual system has different photoreceptor cells, namely the rods, to sense intensities of light (luminance, or luma), and the cones, to sense colors (chrominance, or chroma). As the human visual system is less sensitive to color than to luminance, it is possible to sub-sample the color pixels.

Common color sub-sampling rates are 4:2:0, in which for each four luma samples, there are only one Cb sample and one Cr sample. It actually represents 4:1:1 ratio, but the literature defined as 4:2:0. Other chroma sub-sampling ratios are 4:2:2 and 4:4:4, the later with no sub-sampling (RICHARDSON, 2010). Color sub-sampling may be considered a first tool for video compression.

2.1.3 Video Quality Metrics

Since video coding standards provides usually lossy compression, to achieve large compression rates, it is important to measure the final video quality after video encoding. Video quality is a complex parameter to define, because it is based on subjective criteria of video observers. Subjective metrics, such as Mean Opinion Score (MOS) exist and are used to evaluate the video quality to compare different video coding standards. However, MOS cannot be used in all contexts, because in some cases it is needed to compare and evaluate videos faster and objectively.

Peak Signal-to-Noise Ratio (PSNR) is a well-known objective video quality metric used in the literature (GHANBARI, 2003). PSNR is defined in Equation 2.1, in which MAX is the maximum value of representation of one sample ($2^n - 1$, for 8 bits, MAX is equal to 255) and Mean Squared Error (MSE) is calculated as shown in Equation 2.2. In Equations 2.1 and 2.2, m and n are the horizontal and vertical dimensions of the picture. In Equation 2.2, O and R are the original and reconstructed pictures, respectively. Reconstruced picture is the picture after the coding losses.

$$\mathrm{PSNR(dB)} = 20 \cdot \log_{10}\left(\frac{\mathrm{MAX}}{\sqrt{\mathrm{MSE}}}\right) \tag{2.1}$$

$$\mathrm{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(R_{i,j} - O_{i,j}\right)^2 \tag{2.2}$$

Often it is useful to compare video quality between two different coded videos (e.g. using different codecs) of a same input raw video. Two different coded videos of a same input raw video may incur in different PSNR, but also different bit rate (the rate of a coded video bits, in bits/s). In this case, a simple PSNR comparison is not useful, because videos have also different bit rate values. In this situation, the Bjontegaard Delta PSNR (BD-PSNR) metric (BJONTEGAARD, 2001) must be used. This metrics is based on curve fitting of two different Rate Distortion (RD) curves (of the two different coded videos) formed by four bit rate/PSNR points. BD-PSNR represents an average difference of PSNR values (in dB) over the range of four bit rates. BD-Rate metric also exists and represents an average bit rate difference (in %) over the range of four PSNR values (BJONTEGAARD, 2001). More details about BD-PSNR and BD-Rate calculation can be found in (BJONTEGAARD, 2001).

2.2 Video Coding Background

2.2.1 Brief History of Video Coding Standardization

The history of video coding standardization can be summarized by revising the video coding standards developed by ITU and ISO organizations. The first video coding standard is H.261 (ITU, 1990). Three years later, ISO produced its first video coding standard, MPEG-1 (ISO, 1993). One year later, ITU and ISO groups jointly produced the H.262/MPEG-2 video coding standard (ITU and ISO, 1994). In 1995, ITU produced the H.263 standard (ITU, 1995). ISO produced MPEG-4 Visual standard in 1999 (ISO, 1999). After that, ITU and ISO jointly produced the H.264/MPEG-4 Advanced Video Coding (AVC) standard (also known as H.264/AVC) (ITU and ISO, 2003).

H.264/AVC is the current video coding standard in use in the market, targeting many video applications and ranging different video resolutions. It was developed by the Joint Video Team (JVT) formed by experts of both ITU and ISO. H.264/AVC

doubles the video compression when compared to MPEG-2 standard for similar video quality (WIEGAND, 2003).

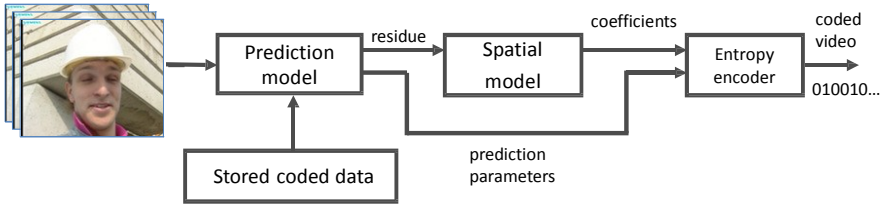
The new High Efficiency Video Coding (HEVC) standard was recently developed by JCT-VC, formed by experts of ITU and ISO, and published in April 2013 (ITU and ISO, 2013). Its primary goal was to double compression efficiency of the most efficient video coding standard until now, namely the H.264/AVC, especially focusing on ultra-high resolution video encoding (beyond 1920x1080 pixels). In recent video coding standards, i.e. H.264/AVC and HEVC, only the video decoder and the bitstream syntax are subject of standardization. The video encoder flow is not standardized. It gives some freedom to design video encoder, because the only requirement for video encoder is to generate a conformed bitstream (WIEGAND, 2003).

2.2.2 Video CODEC

Digital video pictures are compressed by a video encoder that transforms original pictures into a stream of bits of coded video (also called bitstream). To display the coded video, a video decoder must be used to transform the coded video into a reconstructed picture. Usually, the reconstructed pictures are different from original pictures (in lossy compression). When reconstructed and original pictures are equal, it is called a lossless compression (RICHARDSON, 2010). The pair encoder/decoder forms the so-called COder/DECoder (CODEC).

An abstract diagram of video encoder is depicted in Figure 2.1. It is formed by three main functional units: a prediction model, a spatial model and the entropy encoder (RICHARDSON, 2010). The prediction model receives original (raw, uncompressed) pictures and reduces the spatial and temporal redundancies present in video, by exploiting stored coded data, e.g. neighboring pictures or the same picture. The difference of prediction result and the original picture is called residue. The residue is processed by a spatial model that transforms residue into coefficients, exploiting similarities in the residual picture to reduce spatial redundancy. Prediction and Spatial model parameters are further processed by entropy encoder, which is a lossless encoder that removes statistical redundancy in the data.

Figure 2.1 – Abstract system diagram of video encoder



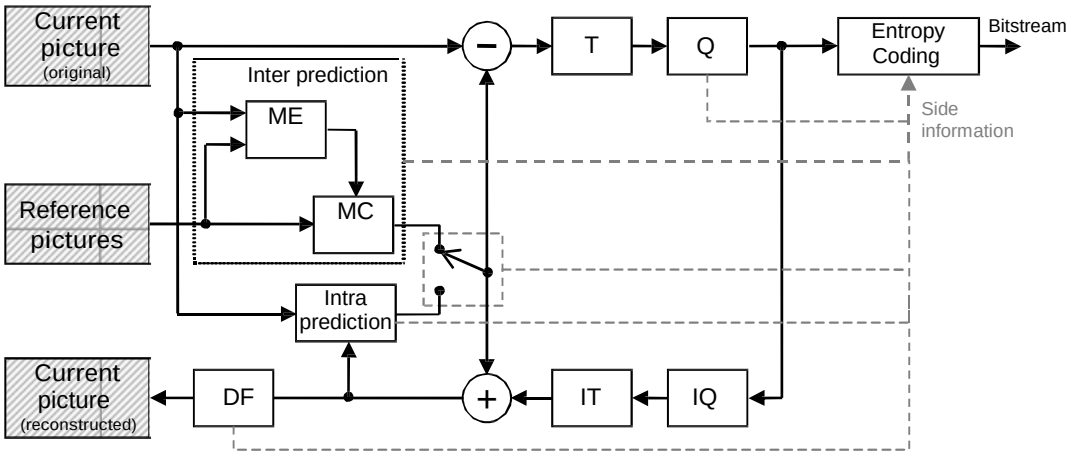
Source: the author, modified from (RICHARDSON, 2010).

Advanced video coding standards, e.g. H.264/AVC and HEVC, follows basically the same CODEC structure. The video encoder is based on the abstract video encoder system diagram shown in Figure 2.1. However, advanced video coding standard are block-based, i.e. video pictures are partitioned into smaller blocks that are processed by the encoder and the decoder. H.264/AVC splits pictures into blocks of 16x16 pixels called macroblocks. The macroblock is the main coding unit that is processed by all the

coding tools of video encoder. HEVC supports larger blocks of size up to 64x64 pixels (SULLIVAN, 2012). More details of HEVC are given in section 2.3.

Figure 2.2 shows the diagram of a generic video encoder. It includes Inter and Intra predictions, Forward Transform (T), Quantization (Q), Inverse Transforms (IT), Inverse Quantization (IQ), Entropy Coding and Deblocking Filter (DF). Intra prediction generates a prediction of a block based on the information of neighboring blocks in the current picture (captured by the camera) that are already processed and reconstructed. Inter prediction is formed by Motion Estimation (ME) and Motion Compensation (MC) coding tools. The goal of Inter prediction is to generate a prediction based on previous pictures, so-called reference pictures. ME searches for the best match, i.e. the block in the reference pictures that is the most similar to the block in the current picture. The most similar block is chosen by ME as the best block, and a motion vector indicates the displacement between current block position and the position of the best block. ME/MC process is detailed in section 2.2.2.1.

Figure 2.2 – System diagram of video encoder



Source: the author, modified from (AGOSTINI, 2007).

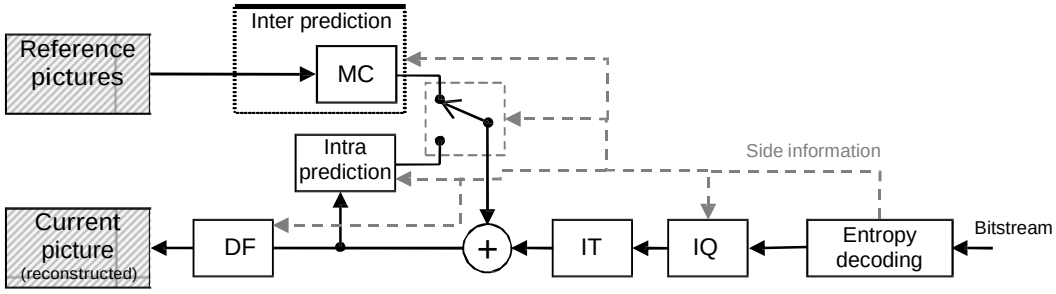
After the prediction, the residual blocks (difference between predicted blocks and original blocks from the current picture) are processed by Transforms and Quantization modules. In H.264/AVC, transform module includes a 4x4/8x8 Discrete Cosine Transforms (DCT) and 2x2/4x4 Hadamard Transform, depending on the residues and the block type (WIEGAND, 2003). Quantization is the module that produces coding losses in the residual blocks, controlled by a Quantization Parameter (QP). QP value is directly proportional to the strength of coding loss. All coding information (residual blocks and side information from other modules, e.g. prediction types, motion vectors, etc.) is further encoded by the Entropy coding module. Advanced video coding standards support sophisticated lossless entropy coding algorithms such as Context Adaptive Binary Arithmetic Coding (CABAC). In particular, H.264/AVC also includes Context Adaptive Variable Length Coding (CAVLC) for entropy coding. The output of entropy coding module is the coded video (bitstream).

Prediction, Transforms, Quantization, and Entropy coding form the forward path of video encoder. The inverse path is formed by Inverse Transforms (IT), Inverse

Quantization (IQ) and Deblocking Filter (DF). The inverse path is included in video encoder to avoid mismatch between encoder and decoder. Reconstructed pictures in video encoder, further used as reference pictures, must match exactly the reconstructed pictures in video decoder, because decoder also uses the reference pictures to perform Motion Compensation (MC). MC is the inverse process of motion estimation. The result of MC or Intra prediction is added to the output of IT to reconstruct the block. Before picture reconstruction, a Deblocking Filter (DF) is applied to remove blocking artifacts caused by strong quantization.

The general diagram of a video decoder is shown in Figure 2.3. It is similar to the inverse path of the video encoder. The coded video (bitstream) inputs to an entropy decoding module. The residual blocks are processed by Inverse Transforms and Inverse Quantization modules. Decoded side information e.g. prediction types, motion vectors, feeds other modules. The result of prediction (either Intra or Inter) is added to the output of Inverse Transforms and is processed by deblocking filter (DF in Figure 2.3) to reconstruct the picture (which will be displayed in the user device). Reconstructed pictures are stored as reference pictures to be used by Motion Compensation.

Figure 2.3 – System diagram of video decoder



Source: the author, modified from (AGOSTINI, 2007).

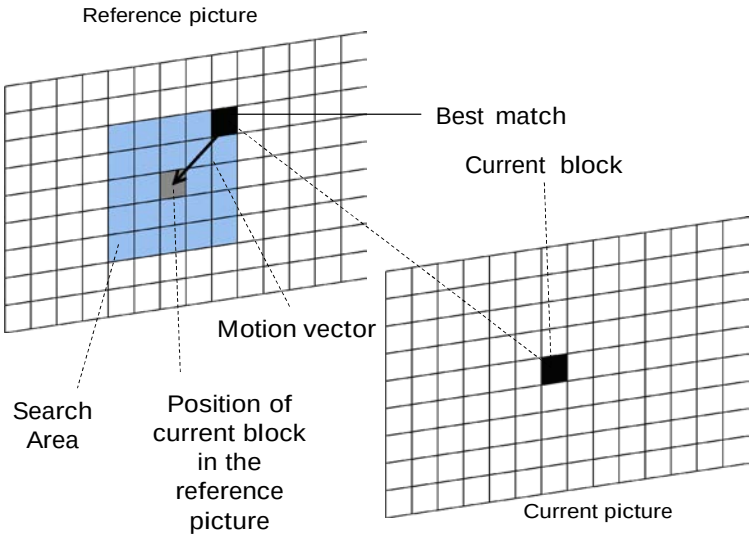
Motion Estimation and Deblocking Filter modules are detailed in sections 2.2.2.1 and 2.2.2.2, since they are the focus of hardware accelerators proposed in this work.

2.2.2.1 Motion Estimation (ME)

Motion Estimation is available only in video encoders, in the Inter prediction module. Figure 2.4 shows the Motion Estimation process. It has the goal to search in the references pictures which block is the most similar with the current block (to be encoded) in the current picture, the so-called best match.

ME search is usually limited to a search area in the reference picture. When the best match is found, a Motion Vector (MV) is generated to indicate the displacement of current block position and the selected block position in the reference picture. The MV is encoded by an entropy encoder and is sent to the bitstream, along with the residue between the best block and the current block, which is transformed and quantized. The ME search is conducted and guided by a block-matching algorithm, which is usually optimized according to specific coding efficiency goals set by the designer (performance, video quality, power, or area, for instance).

Figure 2.4 – Motion estimation process



Source: the author, modified from (PORTO, 2008).

A vast number of ME search algorithms were proposed in the literature, since they are a non-standardized coding tool. A survey of ME algorithms is given in (HUANG, 2006). Independent of the search algorithm, they need to use a video quality metric to determine the best match. PSNR metric, based on MSE, is very complex to be used in practice. Sum of Squared Differences (SSD) is a video quality metric with good correlation with MSE and PSNR. However, calculating square values also requires huge computational effort in real systems. Some video quality metrics that require lower computational effort are the Sum of Absolute Transformed Differences (SATD) and the Sum of Absolute Differences (SAD). SAD is the simpler and requires the lowest computational effort. SATD is based on transformed differences and requires usually a Hadamard transform (RICHARDSON, 2010).

SAD calculation, used in this work, is shown in Equation 2.3, where m and n are the dimensions of the block in samples (horizontal and vertical), O is the current (original) block and R is the reference block.

$$SAD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left| R_{i,j} - O_{i,j} \right| \tag{2.3}$$

One SAD is calculated for one current block and for one specific block out of the many possible block candidates in the reference picture, as determined by the ME search algorithm. The SAD metric is then often used for low complexity video quality measure in the context of block matching for ME.

In advanced video coding standards, ME has many features, e.g. variable block size, bi-prediction (B pictures), weighted prediction, fractional-pixel motion vectors, etc. Fractional-pixel motion vectors require an Interpolation Filter to calculate the value of the fractional position pixels.

2.2.2.2 Deblocking Filter (DF)

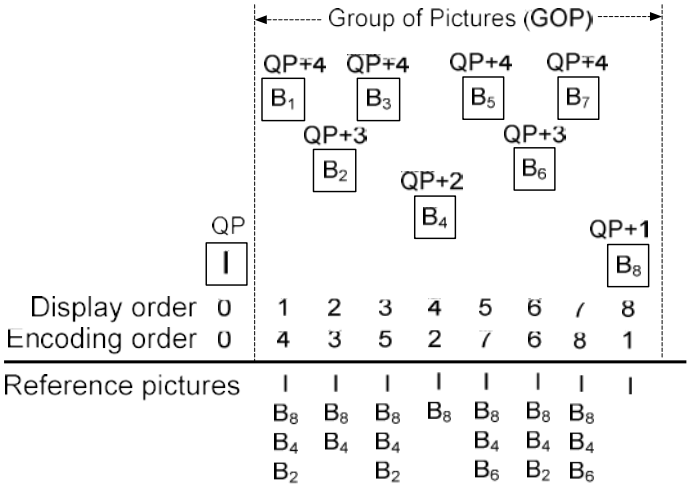
Deblocking Filter is included in the advanced video coding standards to improve the reconstructed picture to be used as reference pictures for ME/MC. It reduces the blocking artifacts present in a reconstructed video, which are introduced by the inherent block partitioning and strong quantization in video encoding. This is done by modifying samples along the vertical and horizontal borders of blocks. The Deblocking filter is adaptive to identify whether the artifacts are real edges of the picture (and thus must not be filtered) or are the edges introduced by the encoding process, which then must be filtered. Therefore, many decisions are applied before the sample modifying to decide whether the filter must be applied or not.

2.2.3 Temporal picture structure

There are two different picture types in video coding standards: Intra (I) picture, which includes only Intra prediction blocks, and Inter picture, that may include Inter and Intra blocks. Regarding Inter prediction, there are two types of pictures: Predictive (P) pictures (only P blocks) which employ Inter prediction with only one reference picture, and Bi-predictive (B) pictures, that may include P and B blocks. Bi-prediction is the Inter prediction in which two reference pictures must be used together to predict the same block. Intra picture is always the first picture in a video sequence, because there are no previous (reference) pictures to perform Inter prediction. It is usual to also include Intra pictures in regular intervals between Inter pictures to provide the so-called Random Access, i.e. for broadcast and streaming video applications, the decoder must find the first Intra picture to start decoding the video stream. The period of pictures between two Intra pictures is referred in this work as intra period.

Both P and B pictures may use reference pictures from the past or from the future of pictures. To support this feature, the encoding order is different to the display order. The display order is often referred as Picture Order Count (POC). The encoding order is called frame number or picture number. In video coding standards, pictures are organized in Group of Pictures (GOP) for encoding. An efficient temporal picture coding structure in terms of compression is the hierarchical B-picture coding structure shown in Figure 2.5. In this structure, pictures are encoded as I pictures or B pictures (B pictures are more generic than P pictures, because they include both P and B blocks).

Figure 2.5 – Temporal picture coding structure using Random Access configuration



Source: the author, modified from (VANNE, 2012).

In the picture structure shown in Figure 2.5, different QP values are assigned for each picture depending on their picture index and type (I or B). Figure 2.5 also shows the prediction dependencies among pictures, i.e. reference pictures (VANNE, 2012).

2.3 Overview of the High Efficiency Video Coding (HEVC) Standard

HEVC is the new video coding standard (ITU and ISO, 2013). It achieves approximately 50% bit rate reduction over H.264/AVC standard (ITU and ISO, 2003). HEVC follows the same block-based video encoder structure as reviewed in section 2.2. To achieve such compression efficiency, HEVC employs larger block sizes (up to 64x64 pixels) and a new flexible quadtree structure that splits those blocks hierarchically down to 4x4-pixel elementary blocks. A new set of advanced coding tools was also introduced in HEVC, as we review in this section.

In HEVC, pictures are first partitioned into Coding Tree Units (CTUs). The size of CTUs is defined by video encoder depending on encoder constraints and video content. CTUs can be as large as 64x64 pixels but may be smaller. CTUs are recursively partitioned into Coding Units (CUs) with a quadtree partitioning structure (each CU in the root may be partitioned into four smaller CUs in the leaves). Hence, HEVC defines three basic units below CTUs:

- Coding Unit (CU): CUs have the size varying from 8x8 pixels to 64x64 pixels. The decision between Intra or Inter-prediction for the unit (or block) PU is taken here. A CTU may contain only one CU or several smaller CUs. Each CU can be individually partitioned into Prediction Units (PUs) and Transform Units (TUs).
- Prediction Unit (PU): defines the type of a prediction block. The largest size of a PU is the CU size in the root. PU sizes range from 4x4 up to 64x64 pixels. They may assume symmetrical and asymmetrical sizes, depending on Intra or Inter prediction. There are 35 intra prediction modes defined in HEVC (33 directional modes, a Planar mode and a DC mode).
- Transform Unit (TU): defines the sizes of transform blocks. Transforms also from their own transform tree, so-called Residual Quad Tree. Transform sizes range from 4x4 pixels to 32x32 pixels. The DCT transform is defined (similar to the ones in H.264/AVC) and a new Discrete Sine Transform (DST) is introduced in the HEVC standard.

Inside the HEVC in-loop filter, a new Sample Adaptive Offset (SAO) filter was introduced, that is applied after the Deblocking Filter. Some tools or schemes originally used in the H.264/AVC were removed, such as the CAVLC entropy coding, and the Hadamard transform. Other tools from H.264/AVC are kept in HEVC, but with some modifications, such as the Fractional-pixel Interpolation Filter, the Deblocking Filter (DF) and the CABAC entropy encoder/decoder.

Some tools were also introduced in HEVC to facilitate the parallel processing of the video coding software. In H.264/AVC, one way to extract parallelism for video coding is by grouping contiguous macroblocks into slices, since slices can be independently encoded. However, the partition of video frames into many slices may introduce high amount of bits for slice header that increases bit rate (because slices were not created for parallel processing, but for robust video transmission). HEVC introduces Tiles, which split the video pictures into rectangular regions that can be encoded in parallel. A

Wavefront Parallel Processing (WPP) scheme was also introduced, which creates a "wave" ordering of macroblocks that can be encoded in parallel, without mutual data dependences.

Table 2.1 summarizes the comparison of H.264/AVC and HEVC coding tools, according to our previous analysis.

Table 2.1 – Comparison of H.264/AVC and HEVC coding tools.

Coding tool	H.264/AVC	HEVC
Size of Coding Unit	16×16 (macroblock)	64×64, 32×32, 16×16, 8×8
Size of Prediction Unit	16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4	64×64, 64×32, 32×64, 32×32, 32×16, 16×32, 16×16, 16× 8, 8×16, 8×8, 8×4, 4×8, 4×4
Size of Transform Unit	8×8, 4×4, 2×2	32×32, 16×16, 8×8, 4×4
Transforms	DCT, Hadamard	DCT, DST
Entropy coding	CABAC, CAVLC	CABAC
In-Loop Filter	Deblocking Filter	Deblocking Filter, SAO
Fractional-pixel Interpolation	6-tap (luma), 2-tap (chroma)	7-/8-tap (luma), 4-tap (chroma)
Intra prediction modes	9 modes (4×4) and 4 modes (16×16, chroma)	35 modes
Parallel processing	Slices	Tiles, WPP

Source: the author.

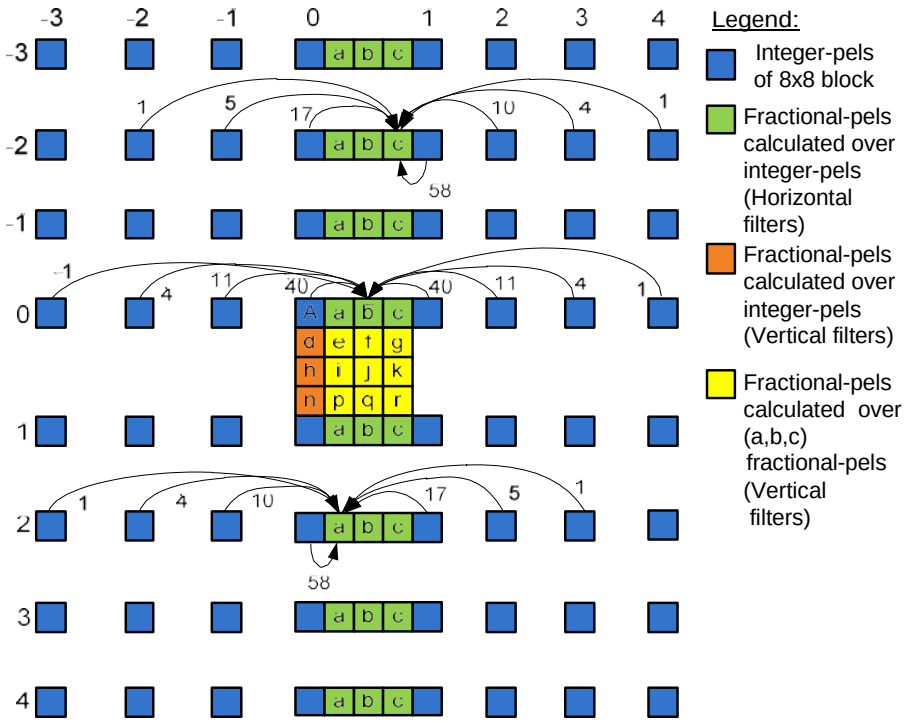
Some coding tools are reviewed in details in this section, such as the Fractional-pixel Interpolation Filter (section 2.3.1) and Deblocking Filter (section 2.3.2), that are the main focus of this work. For more information on other coding tools, please refer to (SULLIVAN, 2012) and (ITU and ISO, 2013).

2.3.1 Fractional-pixel Interpolation Filter

In HEVC, luma and chroma motion vectors (MV) have quarter- and eighth-pel (pixel) precision, respectively. The luma half- and quarter-pels are generated using 8-tap and 7-tap filters, respectively. The chroma eighth-pels are generated using a 4-tap filter. The HEVC interpolation filter improves the coding efficiency of 6-tap and 2-tap (bilinear) interpolation filters of H.264/AVC by 10% (LV, 2012). Along with the Hadamard transform for decision in FME, it reduces the bit rate by 0.3%-2.2% and increases the video quality by 0.04 dB-0.16 dB (CORRÊA, 2012).

Figure 2.6 shows the positions of the luma fractional-pels (green, yellow and orange) with the luma integer-pels (blue). The arrows in Figure 2.6 highlight the integer-pels used for interpolation of a, b and c. The numbers beside the arrows are the filter coefficients. Each luma fractional-pel in green (a, b, or c) is calculated by applying a horizontal filter over the luma integer-pels at positions [-3…4]. The remaining fractional-pels are generated through vertical filters over the integer-pel (orange) or over previously calculated fractional-pels (yellow). There are three different types for luma filters as shown in Table 2.2.

Figure 2.6 – Luma fractional pixel positions for a 8x8 luma integer pixel block



Source: (DINIZ, 2015a).

The luma half-pels (b, h, i, j, and k in Figure 2.6) are generated using an 8-tap filter (filter type 2/4 in Table 2.2) and the quarter-pels are generated using a 7-tap filter (filter types 1/4 or 3/4, in Table 2.2). There are seven types of 4-tap chroma filters (1/8 to 7/8). Table 2.2 shows all the HEVC interpolation filter types and their corresponding coefficients (ITU-T, 2013).

The interpolation filter types and coefficients are the same when used for MC or FME. However, the set of pixels and the order they are calculated may differ. MC interpolates according to the fractional-precision MV received in the bitstream. FME (at the encoder side) has a different behavior. Although FME's process is not standardized, the HM HEVC reference software (HM, 2013) employs the same interpolation filters to generate 8 half-pel points for motion refinements.

Table 2.2 – 7-tap and 8-tap luma and 4-tap chroma filter coefficients.

	Filter type	Filter coefficients							
Luma	1/4	-1	4	-10	58	17	-5	1	
	2/4	-1	4	-11	40	40	-11	4	-1
	3/4		1	-5	17	58	-10	4	-1
Chroma	1/8			-2	58	10	-2		
	2/8			-4	54	16	-2		
	3/8			-6	46	28	-4		
	4/8			-4	36	36	-4		
	5/8			-4	28	46	-6		
	6/8			-2	16	54	-4		
	7/8			-2	10	58	-2		
	Index	-3	-2	-1	0	1	2	3	4

Source: (DINIZ, 2015a).

To illustrate the filter calculation, Equation (2.4) shows an example of $b_{0,0}$ luma half-pel calculation (8-bit depth) with $A_{i,0}$, i=-3..4 integer-pel as input.

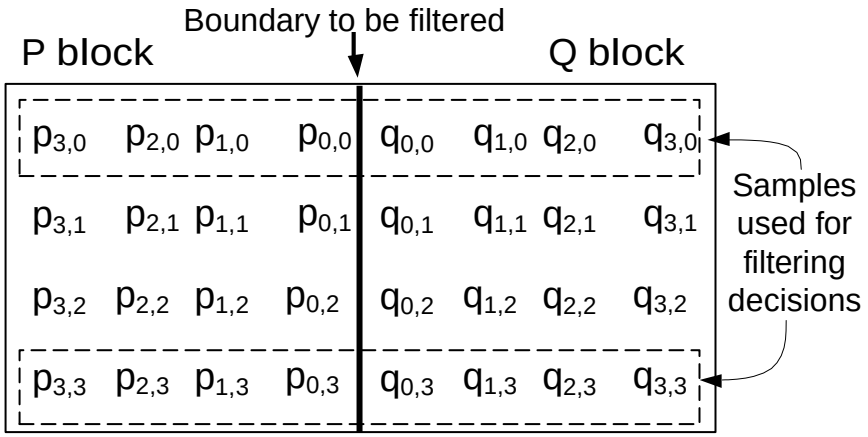
$$b_{0,0} = -A_{-3,0} + 4 * A_{-2,0} - 11 * A_{-1,0} + 40 * A_{0,0} + 40 * A_{1,0} - 11 * A_{2,0} + 4 * A_{3,0} - A_{4,0} \quad (2.4)$$

2.3.2 Deblocking Filter (DF)

The new DF in HEVC contributes to up to 6% bit-rate reduction (1.3%-3.3% bit-rate reduction on average) for the same video quality (NORKIN, 2012). Although the DF is an optional feature in video encoder (and thus may not be used in video decoder), it is often employed because of the high bit-rate reduction.

The deblocking filter reduces the blocking artifacts (visible discontinuities in the video) caused by block-based encoding with strong quantization. It is applied by modifying samples along horizontal and vertical boundaries of PUs and TUs of size not smaller than 8x8 samples. Filtering is applied separately in 4x4 blocks (so-called P and Q blocks), as shown in Figure 2.7. Normal and strong filtering modes modify 2 and 3 luma samples along each boundary, respectively. The example in Figure 2.7 shows a vertical boundary, as the horizontal boundary filtering is analogous.

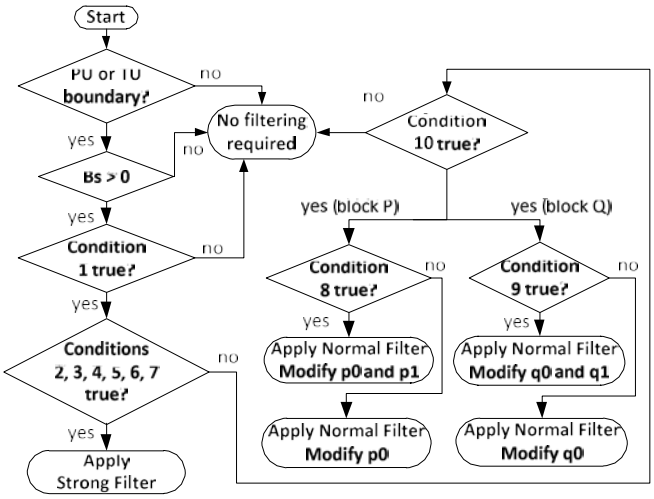
Figure 2.7 – Boundary of a 4x4 block (blocks P and Q)



Source: (DINIZ, 2015b).

Before the boundary filtering, many filtering decisions have to be assessed to decide whether or not the boundary should be filtered, and also to determine which filtering modes, i.e. normal or strong, must be applied on a boundary. Only the samples in the first and last rows of P and Q blocks are used for filtering decisions. Figure 2.8 shows the complete algorithmic flow of the deblocking filter, along with the filtering decision equations.

Figure 2.8 – HEVC Deblocking filter flow and filtering decision equations



Filtering decisions (conditions):

$$|p_{2,0} - 2p_{1,0} + p_{0,0}| + |p_{2,3} - 2p_{1,3} + p_{0,3}| + |q_{2,0} - 2q_{1,0} + q_{0,0}| + |q_{2,3} - 2q_{1,3} + q_{0,3}| > \beta \quad (1)$$

$$|p_{2,i} - 2p_{1,i} + p_{0,i}| + |q_{2,i} - 2q_{1,i} + q_{0,i}| < \beta/8 \quad i=0\ (2),\ i=3\ (3)$$

$$|p_{3,i} - p_{0,i}| + |q_{0,i} - q_{3,i}| < \beta/8 \quad i=0\ (4),\ i=3\ (5)$$

$$|p_{0,i} - q_{0,i}| < 2.5t_c \quad i=0\ (6),\ i=3\ (7)$$

$$|p_{2,0} - 2p_{1,0} + p_{0,0}| + |p_{2,3} - 2p_{1,3} + p_{0,3}| < 3/16\beta \quad (8)$$

$$|q_{2,0} - 2q_{1,0} + q_{0,0}| + |q_{2,3} - 2q_{1,3} + q_{0,3}| < 3/16\beta \quad (9)$$

$$|\delta_0| < 10tc \quad i=0..3 \quad (10)$$

Source: (DINIZ, 2015b).

Filtering decisions avoids the filtering of real video boundaries, filtering only those which are artificially generated by the coding process. Filtering decisions depend upon various parameters, such as block type, QP, and video content. β and t_c value are determined by a lookup table with QP as input, as shown in Table 2.3.

Table 2.3 – Derivation of threshold variables β and t_c for each QP.

QP	0	...	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
β	0	…	6	7	8	9	10	11	12	13	14	15	16	17	18	20	22	24	26	28
t_c	0	..	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3
QP	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
β	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62	64	-	-
t_c	3	4	4	4	5	5	6	6	7	8	9	40	11	13	14	16	18	20	22	24

Source: (ITU and ISO, 2013).

Block types affect boundary strength (Bs) (ITU and ISO, 2013). Chroma filter is only performed when Bs is equal to 2 and no further decisions need to be evaluated. Only the samples closer to the block boundary are modified.

After the filtering decision flow, normal or strong filtering may be applied to modify samples along the boundary. If the normal filter must be applied for a boundary, and

only p_0 and q_0 samples must be modified (see Figure 2.8), the Equations (2.5) and (2.6) are applied. In Equations (2.5) and (2.6), p_0, Δ_0, p'_1, and q'_0 are the unfiltered and filtered samples of P and Q blocks, respectively. Δ_0 is obtained by clipping $_0$, which is shown in Equation (2.7). If the samples p_1 and q_1 must also be modified according to the decision flow, the equations (2.8) and (2.9) are applied, where p_1, q_1, p'_1, and q'_1 are the unfiltered and filtered samples of P and Q blocks, respectively. Δ_{p1} and Δ_{q1} are obtained by clipping $_{p1}$ and $_1$, shown in Equations (2.10) and (2.11), respectively.

$$p'_0 = p_0 + \Delta_0 \tag{2.5}$$

$$'_0 = _0 - \Delta_0 \tag{2.6}$$

$$_0 = (9(_0 - p_0) - 3(_1 - p_1) + 8) \gg 4 \tag{2.7}$$

$$p'_1 = p_1 + \Delta_{p1} \tag{2.8}$$

$$'_1 = _1 + \Delta_1 \tag{2.9}$$

$$_{p1} = (p_2 + p_0 + 1) \gg 1 - p_1 + \Delta_0 \gg 1 \tag{2.10}$$

$$_1 = (_2 + _0 + 1) \gg 1 - _1 - \Delta_0 \gg 1 \tag{2.11}$$

Strong filter is similar to the normal filter. Offset values Δ_0, Δ_1 and Δ_2 are obtained after clipping $_{0s}$, $_{1s}$ and $_{2s}$, which are shown in Equations (2.12), (2.13) and (2.14). Offset values are added to the unfiltered samples p_0, p_1 and p_2 to determine the filtered samples p'_0, p'_1, and p'_2. Samples of Q block are obtained with the same equations, by exchanging p by q. Chroma deblocking filtering modifies only samples p_0 and p_1. The chroma offset calculation is shown in Equation (2.15).

$$_{0s} = (p_2 + 2p_1 - 6p_0 + 2_0 + _1 + 4) \gg 3 \tag{2.12}$$

$$_{1s} = (p_2 - 3p_1 + p_0 + _0 + 2) \gg 2 \tag{2.13}$$

$$_{2s} = (2p_3 - 5p_2 + p_1 + p_0 + _0 + 4) \gg 3 \tag{2.14}$$

$$_c = (p_0 - _0) \ll 2 + p_1 - _1 + 4 \gg 3 \tag{2.15}$$

Further details of HEVC deblocking filter are given in (ITU and ISO, 2013) and (NORKIN, 2012).

2.3.3 HEVC reference software and common test conditions

HEVC Model (HM) (HM, 2013) is the reference software that implements the conforming HEVC video encoder and decoder. It is free to download. One important document is the Common Test Conditions document (BOSSEN, 2013) which defines standard video sequences along with their pictures rates, picture count and resolution. It also defines some coding configurations, i.e. All Intra, Random Acess, and Low Delay, that provide difference temporal picture structures.

2.4 Reconfigurable Computing Background

Reconfigurable computing is the field that develops circuits which are first fabricated but may be programmed in the field, i.e. after silicon fabrication. They can be

classified as Fine-grained (FG) or Coarse-Grained (CG) reconfigurable fabrics, depending on the granularity of the basic reconfigurable hardware element.

The most popular FG reconfigurable fabric in the market today is the Field-Programmable Gate Array (FPGA). Xilinx invented the first commercially viable FPGA in 1985. FPGA is an array of basic Look-up Tables (LUTs) connected together as Configurable Logic Blocks (CLBs). A LUT can usually implement an arbitrary 4-input logic function defined by the user in configurable memory, usually a Random Access Memory (RAM). The major commercial FPGA providers are the companies Xilinx and Altera. Recent FPGAs implement 5-input, 6-input or even 8-input LUTs, and include many more features, like: embedded Block RAM (BRAM) blocks, DSP or multiplier hardwired blocks, hardwired custom blocks, processor cores, etc. Advanced FPGAs also support Partial Run-Time Reconfiguration (PRR) regions (XILINX, 2010) (ALTERA, 2010). PRR is a feature by which the FPGA user can reconfigure regions of FPGA while the other regions are in execution.

In the literature one can find a vast number of CG reconfigurable fabrics. They are designed by coupling a programmable Arithmetic Logic Unit (ALU) with a context memory and a register file to deal with word-level operations (e.g. 16 bits, 32 bits). A survey of CG reconfigurable fabrics can be found in (HARTENSTEIN, 2001).

A recent trend in reconfigurable computing is the design of mixed-grained reconfigurable processors that integrate several CG and FG reconfigurable elements. They are typically organized in a tiled fashion, on a single chip, to ensure scalability to many cores. A few examples are: MORPHEUS (THOMA, 2003), 4S (SMIT, 2005), and KAHRISMA (KÖNIG, 2010). The FG-reconfigurable elements (such as embedded FPGAs) are more suitable to accelerate bit-level and control-flow operations. However, it is costly in terms of delay, area and power (primarily due to the interconnect elements) compared to a customized hardware (IAN, 2007) (SHANG, 2002). In contrast, the CG-reconfigurable elements accelerate word-level data-flow operations with relatively reduced reconfiguration overhead compared to the FG (due to fewer interconnects and ALU blocks). However, CG-reconfigurable elements suffer from inefficient area utilization and performance loss to accelerate bit-level and control-flow computation. Therefore, these many-core mixed-grained reconfigurable architectures overcome the limitations of individually employing FG- and CG-reconfigurable elements and achieve better performance for applications with heterogeneous processing behavior (i.e., with both control- and data-flow) (KÖNIG, 2010). Multiple concurrently executing tasks on a many-core processor may also be accelerated by sharing a mixed-grained reconfigurable architecture (WATKINS, 2010).

Many-core mixed-grained reconfigurable architectures require a run-time system to adapt to the varying application requirements (e.g., performance, resource demand) and to accordingly manage the reconfigurations. Such a run-time system performs the following four key operations. (1) Application Task Allocation determines which share of the reconfigurable fabric is given for each core of a many-core processor to accelerate their tasks (AHMED, 2011)(SHAFIQUE, 2011)(CHEN, 2012). (2) Custom Instruction and Accelerator Selection chooses a particular implementation version at run time from a set of design-time developed implementation versions for each Custom Instruction (called by the concurrently executing tasks). Each custom instruction consist of several datapaths that can be reconfigured on FG reconfigurable elements (BAUER, 2008), CG reconfigurable elements, or the combination of both (AHMED, 2011b). (3) Accelerator Binding (also referred as online synthesis, online placement or dynamic

placement) determines which datapath of a custom instruction should be placed and reconfigured on which specific FG- and/or CG- reconfigurable element of the mixed-grained reconfigurable architecture and how the interconnections between different datapaths of the same custom instruction will be established. (4) Reconfiguration Scheduling determines the reconfiguration sequence of different datapaths, since typically only one reconfiguration interface is available (BAUER, 2008b).

2.5 Power Dissipation in CMOS

One of the main reasons to design specialized hardware accelerators for systems-on-chip (SoC) is that they are more energy efficient than general purpose processors doing the same function (HAMMED, 2011). The total energy consumption E_{total} of a certain computational kernel is given in Eq. (2.5), where P_{total} is the total power dissipated by the circuit that implements this computational kernel, and t_0-t_1 is the interval of time to complete the computation. Energy consumption of a computational kernel reduces when the time to complete the task reduces (without proportionally increasing power), or when the total power reduces (without proportionally increasing the execution time), or when both total power and time reduce.

$$E_{total} = \int_{t_o}^{t_1} P_{total}(t)dt \tag{2.5}$$

Total power is the power dissipated by the circuit. In all SoCs, this total power is in fact time-dependent, as different tasks and operations mode execute in a very complex chip, where literally hundreds of blocks are executing in parallel, and even some of them can be switched-off to save energy, to reduce temperature, or simply to adjust to a lack of compute power demand.

At the basic electrical level, considering the basic devices present in CMOS SoCs, one may classify the power dissipation according to its basic electrical origin at the logic gate and transistor level. At this basic abstraction level, there are three main power dissipation sources when considering CMOS circuits: leakage power, switching power, and short circuit power.

Leakage power is a common engineering name adopted by practitioners to refer to the CMOS circuit static power. That is, when no digital signal is switching, and the circuit inputs are all static, a residual DC current flows through each and every digital static CMOS gate. The dynamic power is analyzed as being composed by two components, both present when the digital signals are switching: the capacitance switching power and the short circuit power. Eq. (2.6) shows the total power P_{total} as the addition of those three power dissipation sources. The leakage power dissipation, shown in Eq. (2.7), is a result of the leakage current $I_{leakage}$. This DC power also depends on the supply voltage V_{DD}. The leakage current is then an unavoidable current that appears from drain to source through MOS transistors operating in the sub-threshold, and from metal gate electrodes into the channel, drain, and source through the carrier tunneling through insulators. Additionally, the DC leakage has contribution from reverse-biased pn junctions which are inherent to source or drain junctions in bulk CMOS technologies.

The power dissipation contribution from switching capacitors is obviously caused by the charging and discharging of capacitors associated to every signal line present in the digital CMOS circuit. These events only occur when transistors switch the output nodes

of the gate, or even the internal electrical nodes of the logic gates. Eq. (2.8) shows that switching power depends on the load capacitance C_L, the frequency of operation f, the supply voltage V_{DD}, and on the switching activity (α) of any given logic gate output loaded by C_L. Short circuit power, shown in Eq. (2.9), is caused by the short circuit current I_{short} that appears when both P-type and N-type transistor networks are on. The short circuit power also depends on an activity factor (beta), and it depends linearly on the switching frequency f, and on the supply voltage V_{DD} (COSTA, 2000).

$$P_{total} = P_{leakage} + P_{s\ \ tch\ \ g} + P_{short} \tag{2.6}$$

$$P_{leakage} = {}_{leakage} * V_{DD} \tag{2.7}$$

$$P_{s\ \ tch\ \ g} = \frac{1}{2} a * \mathbf{f} * C_L * V_{DD}^2 \tag{2.8}$$

$$P_{short} = f3 * \mathbf{f} * {}_{short} * V_{DD} \tag{2.9}$$

It is possible to reduce both dynamic and static power of hardware designs. Switching power may be reduced by reducing operation frequency, switching activity, area (that reduces the electrical parasitics and the load capacitance C_L) and the supply voltage V_{DD}. However, reducing operation frequency and supply voltage certainly incur in the penalty of increased computation time, and may not reduce energy consumption in the end. Usually, a coordinated variation of applied supply voltage, as well as frequency of operation, can reduce the power consumption of CMOS by orders of magnitude (STANGHERLIN, 2013). However, for most portable applications, the total energy spent on the computation is more relevant than the average power over specific operation modes of the SoC. The most daunting design challenge is in fact to design circuits with techniques that reduce power consumption without compromising the application-required computation time. Static power may be reduced by using some circuit techniques such as power-gating (when the supply voltage is disconnected, that is the CMOS circuit, module, block or the entire SoC is powered-off). Many technology advancements, like using higher dielectric constant insulators for transistor gate electrode insulation, have been introduced in order to reduce sources of static dissipation like the tunneling from the gate to the channel or the drain of the transistor. One strategy to minimize even further the DC off-state current in MOS transistors is to use higher-threshold field-effect transistors – the reason why digital logic gates can be made of a variety of transistors of the same type (N or PMOS), with different threshold voltages (low, standard or high).

2.6 Related Work

2.6.1 Chips and Hardware Accelerators for Video Encoding and Decoding

Due to the high computational complexity, severe performance requirements and energy constrains, many works propose dedicated complete chips for video encoding and decoding. The design of sub-modules of video codec is also subject of research.

An H.264/AVC video encoder chip for quad HDTV is presented in (DING, 2009). An H.264/AVC decoder chip supporting scalable/multiview extensions is presented in (CHUANG, 2010) that provides throughput for single-view 4Kx2K video. A set-top box SoC for free-viewpoint applications supporting 4Kx2K resolution for 9 views is

presented in (TSUNG, 2011). An 8Kx2K decoder chip for H.264/AVC and MVC was introduced in (ZHOU, 2012).

A few works explored FPGA-based designs for sub-modules of HEVC, e.g., intra prediction (KHAN, 2013)(ABRAMOWSKI, 2013), motion estimation (NALLURI, 2013), transforms (DIAS, 2013)(CONCEICAO, 2013), deblocking filter (OZCAN, 2013) and entropy coding (CABAC) (PENG, 2013). A full HEVC encoder or decoder in FPGA was not found in the literature.

2.6.2 Hardware Architectures for Interpolation Filter

In context of the previous generation of video codecs, i.e., H.264/AVC interpolation filter, several works explored hardware architectures focusing on FME and MC. An FME architecture is presented in (CHEN, 2004) that uses 16 multiplier-less datapaths for 6-tap interpolation filter. The work in (WANG, 2007) presents a fast algorithm to reduce the complexity of FME at the cost of a quality loss. A generic reconfigurable architecture for interpolation of different video coding standards is presented in (LU, 2009). In (KAO, 2010), an architecture with three engines is introduced that supports different block sizes along with a resource-sharing scheme. An FME architecture is introduced in (TSUNG, 2009) that replaces the 6-tap filter by a bilinear filter to simplify hardware at the cost of quality loss. An adaptive ME algorithm for H.264/AVC with a hardware architecture is presented in (PASTUSZAK, 2013) that employs an interpolator for FME. The interpolator consumes 6,732 LUTs of Altera Arria II GX FPGA device. The work in (WANG, 2005) presents a hardware architecture for MC along with an approach to reduce the number of interpolation datapaths to 13. The woks in (WANG, 2005a) (LI, 2008) focus on different schemes to reduce the memory data transfers on MC. Interpolation custom instructions for a reconfigurable processor are presented in (SHAFIQUE, 2007) (SHAFIQUE, 2010). The work in (ZATT, 2013) presents an MC hardware architecture for H.264/AVC High 4:2:2 Profile with 12 interpolation datapaths and a cache scheme to reduce memory bandwidth. The aforementioned approaches aim at the previous generation of video codecs, i.e., H.264/AVC and cannot be directly employed for the HEVC due to its different processing nature, operational flow, and different set of filters.

Recently, a few hardware architectures were introduced for the interpolation filtering in HEVC. The work in (GUO, 2012) presents two luma interpolation engines with different throughput along with a scheme for hardware reuse. In (AFONSO, 2013) a simplified FME architecture for FPGAs is presented that processes only 8x8-sized blocks at the cost of a bit rate increase of 13% but does not support Chroma interpolation. Moreover, the above techniques (GUO, 2012) (AFONSO, 2013) are fixed at design-time targeting a worst-case design and cannot adapt its throughput to low-medium complexity scenarios at run-time to achieve better energy efficiency. The complete hardware area (i.e., all accelerators) may not be used all the time in average-case operating scenarios or under changing quality constraints.

2.6.3 Hardware Architectures for Deblocking Filter

Some hardware architectures for the deblocking filter of HEVC were already proposed. Ozcan et al. (OZCAN, 2013) introduced an architecture with 2 datapaths in parallel. Each datapath is configurable to implement all decision and edge filter operations. Operating at 108 MHz clock frequency it is able to encode videos with the 1920x1080 pixels resolution at 30 fps. Shen et al. (SHEN, 2013) proposed a four-stage pipeline architecture. They focus on a new filtering order, but do not provide enough

details of the internal architecture of datapaths for filtering decisions and operations. Operating at 28 MHz clock frequency it is able to encode 4Kx2K video resolution at 30 fps. Shen et al. (SHEN, 2013a) extended the architecture in (SHEN, 2013) to include the sample adaptive offset (SAO) filter in a five-stage pipeline architecture.

2.6.4 Hardware Architectures for Sum of Absolute Differences

Many works are found that include SAD hardware modules, mainly into ME hardware architectures. The work in (LIU, 2007) presents a variable block size full-search motion estimation architecture which employs a 32-parallel SAD tree with 387.2k gates (79% of the total gate count), targeting real-time processing of HDTV 1080p video. The work in (CHANG, 2007) presents a methodology to guide the architectural design for H.264/AVC encoder under resolution and frame rate performance requirements. Integer and fractional motion estimation (IME and FME) parallelism is analyzed using fixed processing unit (PU) SAD hardware, with variation only on the size of SAD array, not on the SAD processing unit. The work in (CHEN, 2006) presents a H.264/AVC encoder chip for 720p with fixed SAD array. The work in (VANNE, 2006) presents a new SAD processing unit and firstly includes a comparison of various SAD processing units in terms on area and delay, but do not address power and energy constraints for low power devices.

2.6.5 Accelerator Binding on Reconfigurable Architectures

Many works propose solutions to the binding problem on fine-grained reconfigurable elements. The works in (WALDER, 2003) (AHMADINIA, 2007) (MARCONI, 2010) formulate the problem as a 2D area partitioning model. They propose methods to identify and maintain free rectangular regions where the accelerators can be reconfigured. Some of them also consider the communication cost and bind datapaths near to each other (AHMADINIA, 2007) (MARCONI, 2010). The 2D area model works well with architectures that contain only fine-grained elements and have homogeneous and rich interconnections between the elements. The work in (GRUDNITSKY, 2012) proposes communication-aware binding schemes to solve communication hazards introduced by fine-grained reconfigurable elements organized in 1D-configuration. The work in (FRIEDMAN, 2009) introduces a tool for compile-time scheduling, placement (based on simulated annealing) and routing for coarse-grained architectures. Compile-time placement cannot react to run-time scenarios unpredictable at compile-time. Such a method is also very complex to be applied at run-time. The work in (JAFRI, 2011) proposes a compression method of configuration data for coarse-grained elements to enable multiple implementation versions that can be selected at run-time. However, no run-time binding scheme is proposed in the work in (JAFRI, 2011). Overall, state-of-the-art accelerator binding schemes do not solve the problem of accelerator binding when jointly considering the fine- and coarse-grained reconfigurable elements in a mixed-grained reconfigurable architecture. Moreover, they do not consider the problem of binding for tile-based reconfigurable processors, which is crucial to enable scalable manycore reconfigurable processors.

3 HIGH EFFICIENCY VIDEO CODING APPLICATION ANALYSIS

Before moving to the details of our proposed hardware accelerators, we discuss in this section the analysis of HEVC encoder and decoder applications. This analysis supports further decisions on which accelerators to design and also the architectural decisions. HEVC application profiling is discussed in section 3.1. A run-time analysis of HEVC application is discussed in section 3.2.

3.1 HEVC Application Profiling

We have profiled the HM software (HM, 2013), that includes both HEVC encoder and decoder implementations, to identify the important computational kernels of the HEVC encoder and decoder. Profiling is performed to quantify the contribution of each coding tool included in HEVC in the total execution time. A similar analysis for the older versions of the HM software can be found in (VANNE, 2012) (CORRÊA, 2012). However, the version of HEVC codec profiled in these works, i.e. HM version 6 (MCCAN, 2012) and HM version 7 (KIM, 2012) no longer correspond to the final standard specification (ITU, 2013). The work in (BOSSEN, 2012) profiled the HM version 8 (KIM, 2012a) for a limited set of Quantization Parameter (QP). In this work we perform an extensive analysis of HM version 10 (KIM 2012b) that conforms to the current standard (ITU, 2013). Our analysis considers different QP values. GNU gprof was used for our analysis (FENLASON, 2000).

3.1.1 Experimental Test Conditions

The experiments are performed using the HM software (HM, 2013) according to the HEVC recommended test conditions document (BOSSEN, 2013). In this work, a subset of the video sequences recommended in (BOSSEN, 2013) were chosen for the HEVC application analysis and for the evaluation of the hardware accelerators proposed in this work. This subset of video sequences is presented in Table 3.1. Video sequences range four video resolutions: 2560x1600 pixels, 1920x1080 pixels, 832x480 pixels and 416x240 pixels.

Table 3.1 – Video sequences used for analysis and evaluation.

Video sequence name	Resolution	Picture Count (frames)	Picture Rate (fps)
Traffic	2560x1600	150	30
PeopleOnStreet	2560x1600	150	30
Nebuta	2560x1600	300	60
SteamLocomotive	2560x1600	300	60
Kimono	1920x1080	240	24
ParkScene	1920x1080	240	24
Cactus	1920x1080	500	50
BQTerrace	1920x1080	600	60
BasketballDrive	1920x1080	500	50
RaceHorses	832x480	300	30
BQMall	832x480	600	60
PartyScene	832x480	500	50
BasketballDrill	832x480	500	50
RaceHorses	416x240	300	30
BQSquare	416x240	600	60
BlowingBubbles	416x240	500	50
BasketballPass	416x240	500	50

Source: the author.

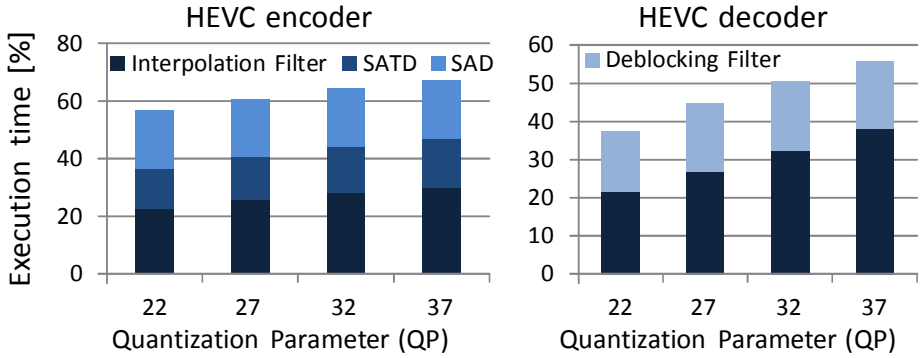
In this work we have employed Random Access (RA) configuration with Group of Pictures (GOP) size of 8. The Intra-period (IP) values are 64, 48, 32, and 24 for pictures rates of 60 fps, 50 fps, 30 fps, and 24 fps, respectively. Each video sequence is encoded using the four recommended QP values (22, 27, 32, 37). The ME algorithm used is TZ Search and the motion search range is 64, which is a default configuration in HM (HM, 2013). The Rate-Distortion Optimized Quantization (RDOQ) feature was disabled, because it is a non-normative feature that increases a lot the computational effort of HEVC encoder and provides less than 4% bit-rate reductions. The increase of computational effort of RDOQ is huge and does not justify the compression efficiency it provides. Other configurations are kept as default in HM (HM, 2013). All experiments were performed on an Intel Core i7-2600 processor with 16 GB memory.

3.1.2 Analysis of HEVC application with an ultra-high resolution video sequence

The first analysis is performed by encoding (and then decoding) the ultra-high resolution PeopleOnStreet video sequence (2560x1600 pixels). We have profiled both HEVC encoder and HEVC decoder application software included in HM (HM, 2013).

Figure 3.1 shows the execution time distribution (in percentage) for the most important HEVC coding tools in terms of computational complexity. In HEVC encoder, 55%-70% of execution time is spent in the following coding tools: Fractional-pixel Interpolation Filter, Sum of Absolute Differences (SAD) and Sum of Absolute Transformed Differences (SATD). The contribution of time of each tool depends on the QP value (given as input by the user) because it influences the mode decision process. As reviewed in Chapter 2, SAD and SATD are distortion metrics used to calculate RD cost for Integer-pixel and Fractional-pixel Motion Estimation (IME/FME), respectively. SATD are also used to calculate RD cost of Intra prediction modes. Interpolation Filter is used to generate the fractional-pixels for FME. Other coding tools, e.g. intra prediction, transforms, quantization, entropy coding, contribute together to the remaining part of execution time in HEVC encoder.

Figure 3.1 – Contribution of different HEVC coding tools (in percentage) to the total execution time. Video sequence: “People on Street” (2560x1600 pixels), 150 pictures



Source: the author.

In the HEVC decoder, 35%-55% of the execution time is spent on Fractional-pixel Interpolation Filter and Deblocking Filter. Interpolation Filter is also required in decoder to reconstruct fractional-pixels during the Motion Compensation (MC) process.

Deblocking filter is the second most complex coding tool in HEVC decoder. Deblocking filter is used to reduce blocking artifacts caused by strong quantization.

Another observation is the time HM software takes to encode and decode this ultra-high resolution video sequence. Encoding 150 frames of this video takes around 26 minutes. Decoding the same amount of frames takes 39 seconds (with QP=37) to 67 seconds (with QP=22).

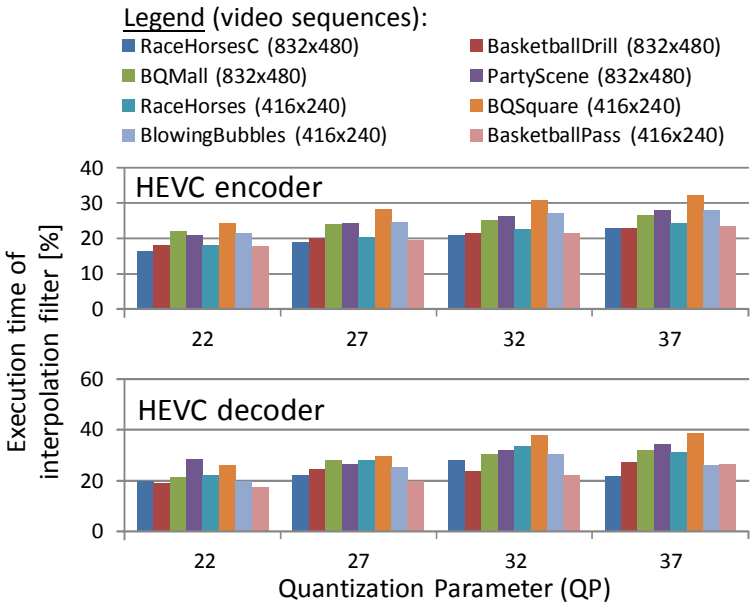
In the next sections, we present a more detailed analysis of the three most complex coding tools: Interpolation Filter, SAD and Deblocking filter.

3.1.3 Analysis of the Interpolation Filter

In this analysis, we have profiled eight video sequences of resolutions 832x480 pixels and 416x240 pixels. The total execution time to encode these videos (10 seconds of video) is 10-106 minutes. Decoding these video takes 1 to 14 seconds.

Figure 3.2 shows the execution time of Interpolation Filter as a percentage of total execution time in the HEVC encoder and decoder. The interpolation filter contributes towards 15%-38% of the execution time in HEVC encoder and decoder depending upon the video sequence and QP.

Figure 3.2 – Contribution of Interpolation Filter (in percentage) to the total execution time of HEVC encoder and decoder for eight video sequences and four QP values



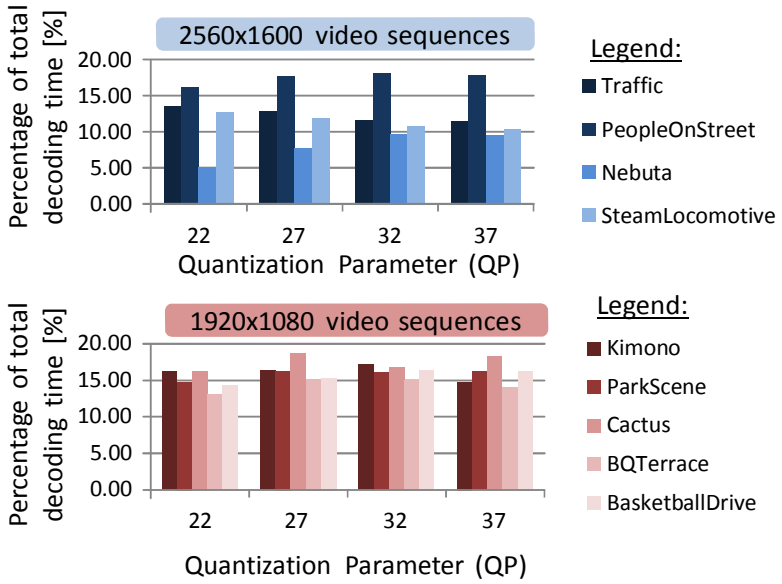
Source: (DINIZ, 2015a).

3.1.4 Analysis of the Deblocking Filter

In this analysis we have profiled only HEVC decoder. Four ultra-high resolution (with 2560x1600 pixels) and five high resolution (with 1920x1080 pixels) encoded video sequences were used as input to the HEVC decoder. Encoded video sequences follow the same coding configuration described in section 3.1.1. Figure 3.3 shows the

execution time of Deblocking Filter as a percentage of total execution time of HEVC decoder. Deblocking filter contributes to 5%-18% to the total decoding time, depending on video sequence and QP. This percentage represents up to 11 seconds spent only in deblocking filter to decode 3 seconds (150 pictures) of 2560x1600 resolution videos. The contribution of deblocking filter in total execution time of HEVC encoder is not significant, since other tools dominate the execution time.

Figure 3.3 – Contribution of Deblocking Filter (in percentage) to the total execution time of HEVC decoder for nine video sequences and four QP values



Source: (DINIZ, 2015b).

3.1.5 Analysis of the Sum of Absolute Differences (SAD) Calculation

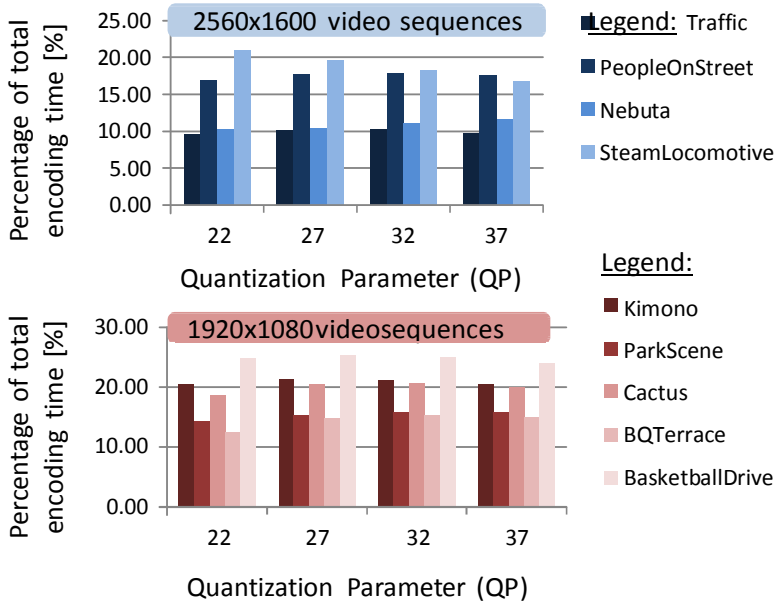
SAD is executed only in the HEVC encoder. It is used to decide the best blocks in Motion Estimation process (i.e. block matching) and Mode Decision, as reviewed in Chapter 2. HEVC decoder does not require SAD, since the block decision is done at encoder side. In this analysis we have profiled only HEVC encoder with four ultra-high resolution (with 2560x1600 pixels) and five high resolution (with 1920x1080 pixels) video sequences. Encoded video sequences follow the same coding configuration described in section 3.1.1.

Figure 3.4 shows the execution time of SAD as a percentage of total execution time of HEVC encoder. SAD contributes to 9%-25% to the total encoding time, depending on video sequence and QP. Therefore, SAD is an important computational kernel in HEVC video encoder.

3.1.6 Summary of HEVC application analysis

By observing the results of our HEVC application analysis, we conclude that the most important computational kernels in the HEVC encoder and decoder are: Interpolation Filter, Deblocking Filter and SAD. It is confirmed by profiling results with various video sequences (of different resolutions) and four QP values. Since the results vary significantly from one video to another and among different QPs, we have also conducted a run-time analysis of HEVC application.

Figure 3.4 – Contribution of Sum of Absolute Differences (in percentage) to the total execution time for various nine video sequences and four QP values



Source: the author.

3.2 Run-time Analysis of HEVC Application

The previous analysis based on software profiling is offline, it does not consider run-time variations. We have chosen the most complex coding tool of both HEVC encoder and decoder, i.e. interpolation filter, to perform a run-time analysis.

We analyze the run-time behavior of the interpolation filter by recording its number of calls per picture (i.e., execution frequency) for each video picture. We monitored the 'filter' method of TComInterpolation class in HM (HM, 2013), which is the basic method for all types of interpolation filter operations defined in HEVC. Figure 3.5 shows the results for the first 180 frames of the BQMall and BasketballDrill sequences encoded with QP=22 and QP=37. The number of filter calls per picture varies from zero (when picture type is Intra, in which Interpolation filter and FME are not executed) to 6 million (see Figure 3.5). Video properties impact considerably the number of calls.

At the decoder side, the number of calls varies due to the variations in the number of fractional-precision MVs in the bitstream. At the encoder side, the number of filter calls varies due to encoder workload variations as a result of the following:

1) The IME/FME is called by a decision process that usually implements an early skip decision, i.e., in which the ME/MC is not performed at the encoder and the MC is performed at the decoder by inferring a MV from the neighbor blocks, already decoded. The early skip decision is dependent on the input QP value and the input video sequence, which cannot be determined at design-time, rather at the run-time. Moreover, the HEVC introduces a quadtree block partitioning structure and the decision process is sensible to this structure. Some regions of video (e.g. highly textured regions) are partitioned into smaller blocks than other regions (e.g. homogeneous regions). In this regions, there are more blocks for IME, and also for FME. Hence, there are more blocks to be interpolated.

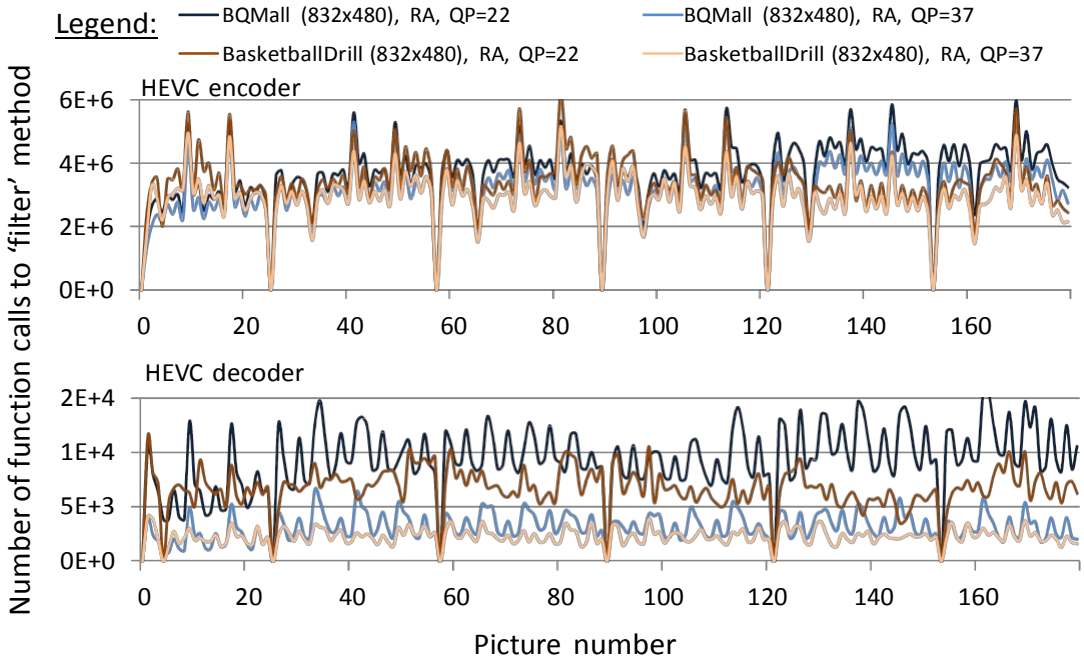
2) In the hierarchical B-picture GOP structure, some pictures perform ME using two or more reference pictures to achieve better quality results. The higher is the number of reference picture, higher is the number of interpolation filter calls. The user that chooses encoding parameters can configure the hierarchical B-picture GOP structure.

3) After the best match is chosen by the FME, the encoder must also perform MC to reconstruct the block correctly to avoid mismatch between encoder and decoder information. In this case, the MC is also performed in the encoder. The number of interpolation filter calls in MC is not determined as it highly depends upon the outcome of the FME and mode decision process.

4) Finally, after the IME/FME is performed for one block, the decision may even decide to encode the block with Intra Prediction. Hence, no interpolation is called in the reconstruction of the block (i.e., MC) inside the encoder.

Because of the above-mentioned differences between the number of calls to the interpolation filter in encoder and decoder, and by looking at our analysis in Figure 3.5, it can be noticed that the number of interpolation filter calls in the encoder is two orders of magnitude higher that the number of interpolation filter calls in the decoder. Moreover, in both encoder and decoder, it depends on many parameters such as video content, prediction type, QP, and video resolution.

Figure 3.5 – Number of calls per picture to the interpolation filter basic method



Source: the author.

In summary, our run-time analysis shows that the number of interpolation filter calls depends upon the video content that cannot be predicted at design-time.

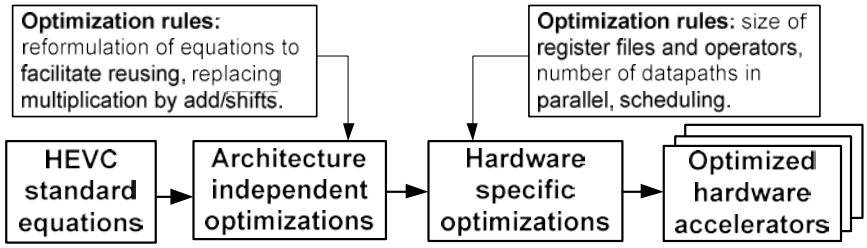
4 DEDICATED HARDWARE ACCELERATORS

This chapter describes the novel dedicated hardware accelerators for important computational kernels of HEVC. First, section 4.1 discusses the methodology to design efficient hardware accelerators. In the following, hardware architectures for Fractional-Pixel Interpolation Filter (section 4.2), Fractional-Pixel Interpolation Filter using Adder Compressors (section 4.3), Deblocking Filter (section 4.4) and Sum of Absolute Differences (section 4.5) are presented and discussed.

4.1 Methodology to Design Hardware Accelerators

This work follows a methodology to design optimized hardware accelerators for HEVC, as shown in Figure 4.1.

Figure 4.1 – Methodology to design optimized hardware accelerators



Source: (DINIZ, 2015b).

First, standard equations from HEVC standard (ITU-T, 2013) are subject to architecture independent optimizations. Examples of these optimizations are the reformulation of equations to facilitate the reuse of many operations using the same hardware operators and the replacing of constant multiplications by add/shift operations (i.e. multiplierless constant multiplication) which are more area and power efficient.

The result of this analysis step contributes to hardware-specific optimizations in the following manner:

1) Determining the sizing of registers and operators: video samples are usually represented in 8 bits data-width. Specific hardware for video does not need to implement 32-bit or 64-bit registers and operators (adders, subtractors, multipliers), which are common in general purpose processors.

2) Determining the number of datapaths in parallel, scheduling, and balancing of operating stages: application throughput requirements must be known to determine the number of datapaths in parallel. Meeting throughput requirements is crucial for real-time video encoding and decoding applications. In video coding applications, throughput is measured in samples (pixels) per second and also in frames (pictures) processed per second of a given picture resolution (e.g. 1920x1080 pixels @ 30 fps). The application requirement may be, for instance, the number of samples to be interpolated in each picture, and the number of pictures per second of a given resolution. Balancing the combinational paths of the accelerators defines the operating frequency, which is also important to meet throughput requirements. Defining the

scheduling of the architecture determines the number of cycles, which also affects throughput. Hence, those three parameters must be considered jointly when designing the architecture of the accelerator.

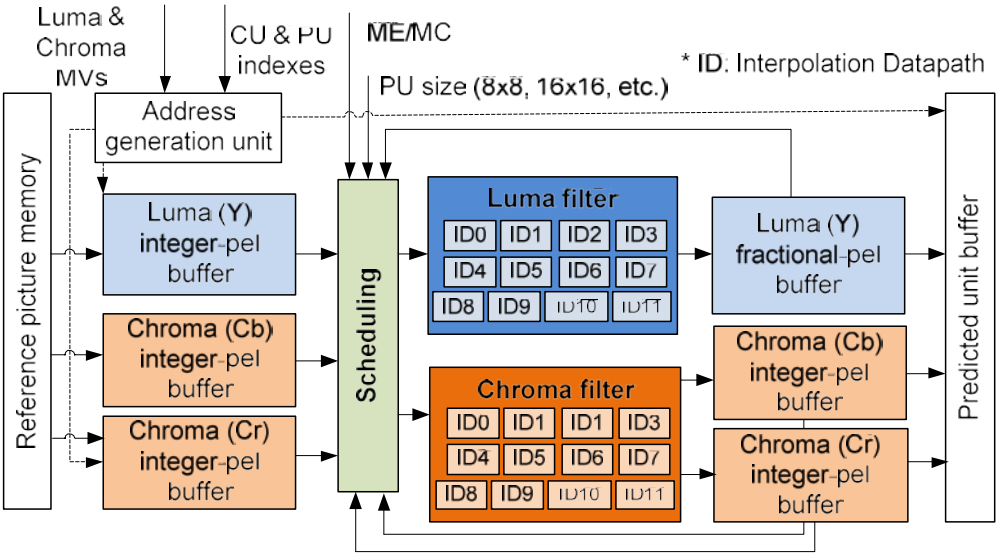
This methodology may be generalized to design accelerators for other dataflow applications that are not in the domain of interest of this work, i.e. video coding. However, the hardware designer may follow this methodology only if the application allows some degree of parallelism (the use of datapaths in parallel), and whether it is possible to apply some of those above-mentioned optimizations. The designer also needs to determine if the application has some throughput requirement, how to measure it, and how to use it to apply the hardware-specific optimizations. Designing accelerators for applications not in the domain of video coding standards is beyond the scope of this thesis.

4.2 Hardware Architecture for Fractional Pixel Interpolation Filter of HEVC

A dedicated hardware architecture for the Fractional Pixel Interpolation Filter is proposed in this work, since this is an important computational kernel in both HEVC encoder and decoder. A generic interpolation filter architecture is proposed to be used in both HEVC encoder and decoder implementations.

The system diagram of the proposed hardware architecture is shown in Figure 4.2. It is composed of luma and chroma filter acceleration engines, integer- and fractional-pel buffers, and a scheduling module to feed the filtering engines. Buffers store integer-pel and fractional-pel that are used for other fractional-pel calculation. In order to meet with the high throughput requirements, each of the acceleration engines for luma and chroma filtering include 12 interpolation datapaths in parallel. With this number of datapaths in parallel the architecture is able to interpolate one line of fractional-pel for 4x4 PU size (it requires 11 integer-pel as input).

Figure 4.2 – System diagram of the proposed hardware architecture for HEVC interpolation filtering



Source: (DINIZ, 2013).

The scheduling module is adaptive to the size of PU used and the execution scenario (ME or MC), to delivers appropriate input pixels for filtering. Other modules that interface with the proposed architecture in a complete system implementation are not the focus of this work. Examples of these modules are reference picture memory to store the reference pictures, predicted buffer to store the output fractional-precision PU, and an address generation unit to store PUs based on corresponding PU and CU indexes and input integer luma/chroma MVs. Sections 4.2.1 and 4.2.2 discuss the internal architecture of luma and chroma filter datapaths, respectively. Section 4.2.3 discusses the adaptive scheduling module in more detail.

4.2.1 Luma Interpolation Filter Datapath

Three filters for luma interpolation were defined in HEVC (ITU-T, 2013): one 8-tap filter for half-pel (2/4 pixel location) and two 7-tap filters for quarter-pel (1/4 and 3/4 pixel locations). In order to reduce area, we designed a configurable datapath that serves for all luma filters. Furthermore, we apply the following design optimizations:

1) 7-tap filters for quarter-pel (1/4 and 3/4) are equal if the input samples are rotated;
2) 8-tap filter for half-pel (2/4) is symmetric, because the first 4 coefficients are equal to the last 4 coefficients, if the last ones are rotated;
3) The coefficients are constant, so the multiplications can be replaced by add/shift operations.

Considering the above three optimizations, only 9 different coefficients must be used to implement 7-tap and 8-tap luma interpolation filtering. Table 4.1 shows a representation of how coefficient multiplication is replaced by an add/shift operations. The goal here is to find a replacement that results in the minimal number of operands.

Table 4.1 – Luma coefficient multiplications replaced by add/shift operations

	Coefficients								
Input samples	1	-1	4	-5	-10	-11	17	40	58
<< 0	+	-		-		-	+		
<< 1					-	-			+
<< 2			+	-					
<< 3					-	-		+	-
<< 4							+		
<< 5								+	
<< 6									+

Source: (DINIZ, 2013).

The following equations (4.1), (4.2) and (4.3) describe the computation needed for our optimized luma interpolation filter. Eq. (4.1) is used for half-pel filter (filter type 2/4). Eq. (4.2) and eq. (4.3) are used for quarter-pel filters (filter types 1/4 and 3/4). These equations support all 9 different coefficients multiplication in a multiplier-less way.

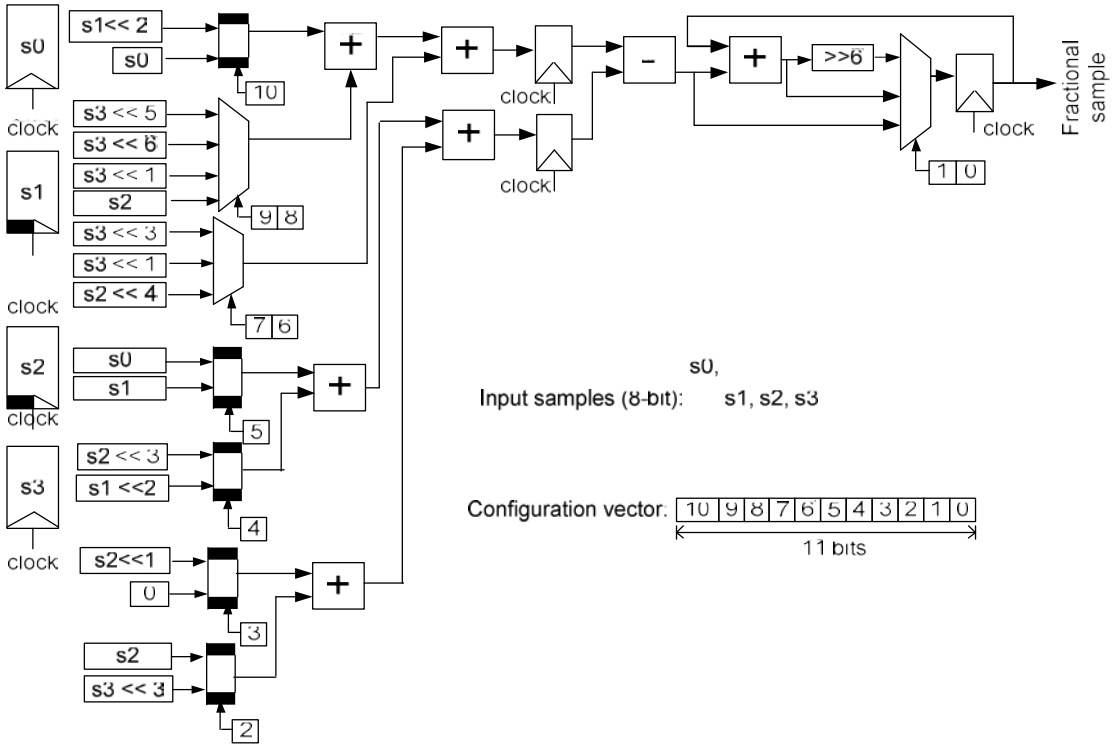
$$h = -1 * s_0 + 4 * s_1 - 11 * s_2 + 40 * s_3 = s_1 \ll 2 + s_3 \ll 5 + s_3 \ll 3 - (s_0 + s_2 \ll 3 + s_2 \ll 1 + s_2) \quad (4.1)$$

$$_1 = -1 * s_0 + 4 * s_1 - 10 * s_2 + 58 * s_3 = s_1 \ll 2 + s_3 \ll 6 + s_3 \ll 1 - (s_0 + s_2 \ll 1 + s_2 \ll 3 + s_3 \ll 3) \quad (4.2)$$

$$_2 = -1 * s_0 + 5 * s_1 + 17 * s_2 = s_0 + s_2 + s_2 \ll 4 - (s_1 + s_1 \ll 2), s_3 \quad (4.3)$$

Figure 4.3 shows the detailed circuit diagram of the configurable datapath for the luma interpolation. Luma filter datapath has four input samples (s_0-s_3) and is divided in a two cycles. The type of filter has to be configured through the configuration vector in order to select one of the three equations (4.1), (4.2) or (4.3). Configuration vector is determined by a Finite State Machine (FSM) inside the scheduling module. Since there are common sub-expressions in those equations, each common sub-expression is implemented only once and the others can be eliminated. With those optimizations, the final luma datapath uses only 7 adders. It represents a 56% saving in adders compared with the work in (GUO, 2012).

Figure 4.3 – Configurable datapath for luma interpolation filter



Source: (DINIZ, 2013).

4.2.2 Chroma Interpolation Filter Datapath

For chroma filter, we applied the same methodology as for the luma filter. Seven different chroma interpolation filter types are defined in HEVC (from 1/8 to 7/8 filter types). Our chroma interpolation datapath supports only 4 filters, since the other 3 filters are equal when the coefficients are rotated. Equations (4.4), (4.5), (4.6) and (4.7) are the computations implemented in our optimized chroma datapath.

$$c_1 = -2 * s_0 + 58 * s_1 + 10 * s_2 - 2 * s_3 = s_1 \ll 6 + s_1 \ll 1 + s_2 \ll 1 + s_2 \ll 3 - (s_0 \ll 1 + s_1 \ll 3 + s_3 \ll 1) \quad (4.4)$$

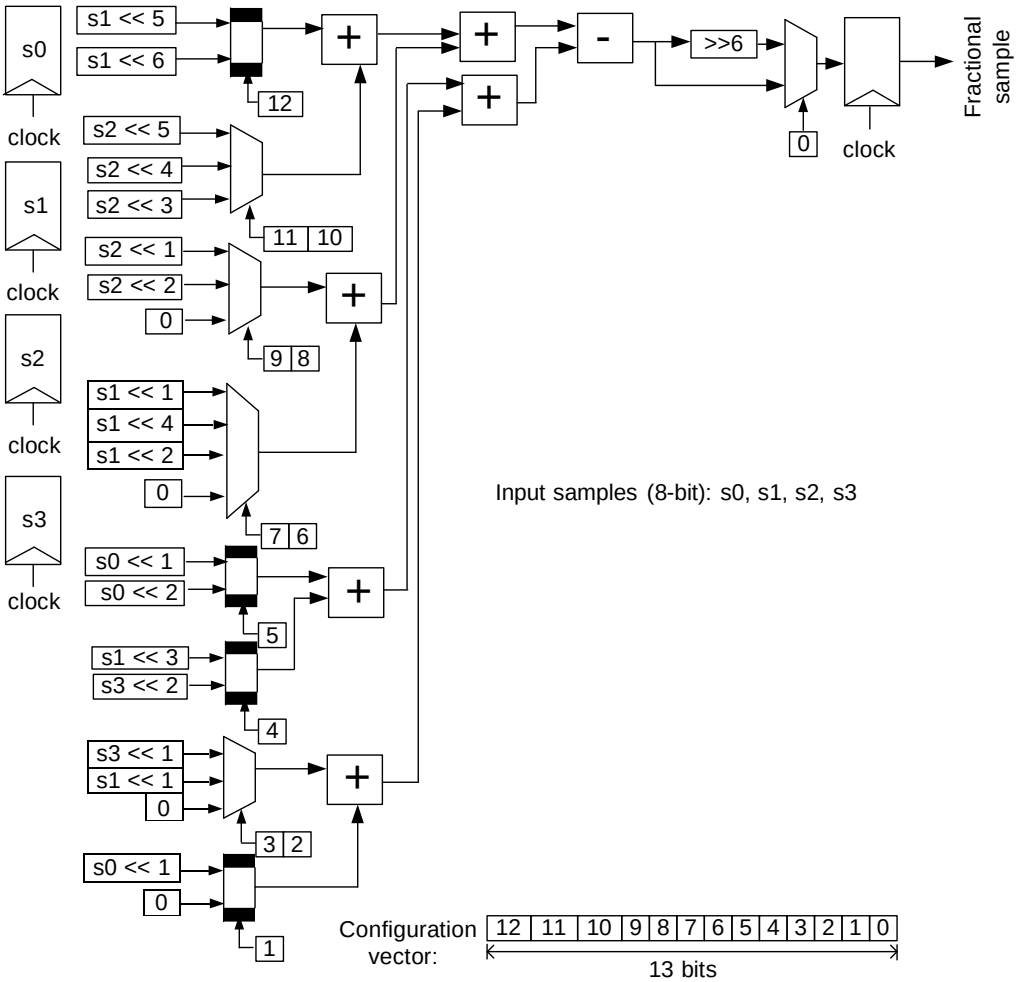
$$c_2 = -4 * s_0 + 54 * s_1 + 16 * s_2 - 2 * s_3 = s_1 \ll 6 + s_2 \ll 4 - (s_0 \ll 2 + s_1 \ll 3 + s_1 \ll 1 + s_3 \ll 1) \quad (4.5)$$

$$c_3 = -6 * s_0 + 46 * s_1 + 28 * s_2 - 4 * s_3 = s_1 \ll 5 + s_1 \ll 4 + s_2 \ll 5 - (s_0 \ll 2 + s_0 \ll 1 + s_1 \ll 1 + s_2 \ll 2 + s_3 \ll 2) \quad (4.6)$$

$$c_4 = -4 * s_0 + 36 * s_1 + 36 * s_2 - 4 * s_3 = s_1 \ll 5 + s_1 \ll 2 + s_2 \ll 5 + s_2 \ll 2 - (s_0 \ll 2 + s_3 \ll 2) \quad (4.7)$$

Since there is no similarity between the coefficients among the 4 different chroma filters, the optimizations applied in luma datapath for reduced area cannot be applied in chroma datapath. The area benefit comes from configurable nature of the datapath. Configuration vector is determined by FSM inside the scheduling module. Figure 4.4 shows our single-cycle configurable datapath supporting four 4-tap chroma filters. The chroma datapath was also designed by removing the common sub-expressions. With those optimizations, it uses only 7 adders.

Figure 4.4 – Configurable datapath for chroma interpolation filter



Source: (DINIZ, 2013).

4.2.3 Scheduling

Interpolation filtering computations for motion estimation and compensation are same, but the order in which the fractional-pels are computed inside a block could be different. FME in HEVC software (HM, 2013) is invoked after IME finds the lowest distortion cost motion vector. Then it interpolates only 8 half-pel points around this

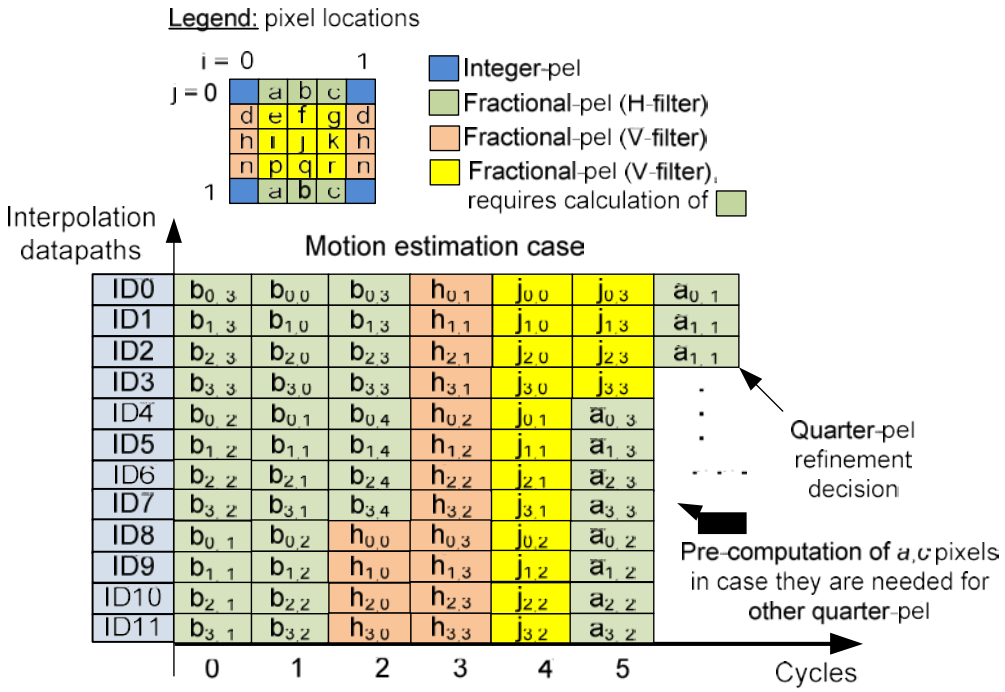
motion vector and refines the search. Another search step is applied after interpolating 8 quarter-pel points around the best half-pel block. The motion vector with the lowest cost in this last search step is selected as the motion vector of the PU (KIM, 2013). This procedure is applied in this way to reduce the number of distortion calculations of FME. In the decoder side, quarter-pel motion compensation must interpolate all fractional-pel inside the PU pointed by the motion vector received in the bitstream.

A different order of half- and quarter-pel computations influences the processing schedule and data fetching from the memory. In case of ME (see Figure 4.5), luma half-pels b, h, and j should be calculated first (see specification of pixel notations in section 2.3.1). We propose a scheduling for our architecture that is parameterized depending on PU size and the usage scenario (ME or MC), which pixels in the same row to be delivered in parallel to the filtering interpolation datapaths. Memory locality is exploited by calculating fractional-pels of the same row first. For example, in our scheduling b pixels are prioritized to be delivered into the 12 luma filter interpolation units, because they are used as input for j pixels calculation.

In case of MC, half- and quarter-pels of the same row (for horizontal filters) and of the same column (for vertical filters) are computed in the sequence such that the memory locality could be further exploited. Each fractional-pel computation in our architecture involves loading the input samples into the luma and chroma interpolation units and configuring their filter types through the configuration vector.

Prediction unit size also influences scheduling regarding memory fetching. For 8-tap filtering, the architecture needs to fetch 8 integer-pels. If PU is of 4x4 size, to exploit all the parallelism, our architecture needs to compute 12 half-pels (e.g. 3 rows of half-pels) and for this 33 integer-pels are fetched. Therefore, our architecture benefits with the increase in PU size, because as more half-pels are calculated in the same row, few integer-pels rows need to be fetched in advance for each interpolation cycle.

Figure 4.5 – Interpolation filter scheduling



Source: (DINIZ, 2013).

This module is implemented as a FSM that also determines the values of configuration vectors for luma and chroma interpolation datapaths.

4.2.4 Results and Evaluation

The proposed interpolation filter architecture was implemented in VHDL and synthesized using TSMC 150nm and FreePDK 45 nm (FREEPDK, 2014) CMOS standard-cell libraries using Cadence RTL Compiler tool (CADENCE, 2014). For both technologies, a constraint of 312 MHz clock frequency was introduced. Synthesis results are shown in Table 4.2.

Table 4.2 – Synthesis results and comparisons to the state of the art hardware implementation of the interpolation filter.

		(GUO, 2012)	GUO, 2012)	Our architecture	
Technology (nm)		90	90	150	45
Gate count	Luma datapath	–	–	1,363	1,298
	Chroma datapath	–	–	1,132	974
	Scheduling/Control	–	–	269	264
	Total (luma only)	19,600	32,496	16,625	15,708
	Total	–	–	30,209	27,396
Memory (#bits)		–	–	1224	1224
Frequency (MHz)		85.5	171	312	312
Throughput		HD1080 @ 30 fps	QFHD @ 60 fps	QFHD @ 30 fps	QFHD @ 30 fps
Power (mW)		–	–	–	23.6

Source: the author.

Our work is compared with two state-of-the-art HEVC interpolation solutions as presented in (GUO, 2012). Our hardware architecture delivers 12-pel at each two cycles (or 6-pel/cycle) for luma interpolation and 12-pel/cycle for chroma interpolation. Operating at 312 MHz frequency, our architecture achieves throughput to support the interpolation processing of Quad Full High Definition (QFHD), i.e. 3840x2160 at 30 fps. Note that compared to the dual-engine architecture of (GUO, 2012), our architecture has half throughput. However, unlike our architecture, the dual-engine architecture of (GUO, 2012) does not support excessive processing for chroma interpolation. Moreover, due to the use of configurable datapaths, our architecture, implemented in 150 nm technology (older technology than related work) requires area reduction of 16% and 49% compared with single- and dual-engine in (GUO, 2012), respectively. The scheduling module allows the use of a reduced input buffer so the memory usage of the architecture is only ≈1Kbit (when PU size is 64x64 pixels).

Regarding the synthesis results for the FreePDK 45 nm CMOS technology, our work also provides an estimation of power dissipation of the architecture proposed herein. Comparing power results to the related work (GUO, 2012) was not possible, since it did not provide any power estimation numbers in the paper.

4.3 Hardware Architecture for Fractional Pixel Interpolation Filter using Adder Compressors

Our hardware architecture for fractional pixel interpolation filter (Section 4.2) is composed by a reasonable number of additions in the filter datapaths. Since the

intermediate addition values are not useful for the application, but only the final interpolated result, we have used multiple-operand, more efficient adder compressors to reduce area and power of the filter datapaths.

A combination of 7-2 and 8-2 adder compressors, which perform the simultaneous addition of 7 and 8 operands respectively are used in the new logic architecture for the interpolation filter architecture. We exploit 7-2 adder compressor and different internal structures of the hierarchical 8-2 adder compressor.

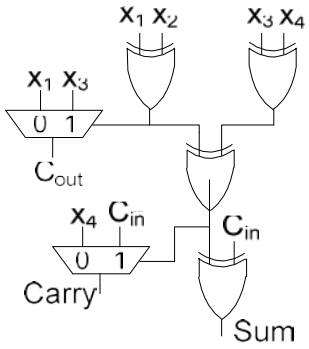
4.3.1 Adder Compressors Background

The focus of this section is not on designing new adder compressor structures, but applying existing adder compressor structures into our proposed fractional pixel interpolation filter architecture. This section reviews some of the existing adder compressor structures available in the literature.

The internal structure of the 4-2 adder compressor is presented in Figure 4.6. It has a reduced critical path since the maximum delay is given by three Exclusive OR (EXOR) gates. The 4-2 compressor has five inputs and three outputs, where the four inputs x_1, x_2, x_3, x_4 and the output Sum have the same weight. On the other hand, the outputs Carry and C_{out} have one bit order higher. One important point to be emphasized in this compressor is the independence of the input carry (C_{in}) in the output carry (C_{out}). This aspect enables implementation of this structure with higher performance. The final sum (S) result of the 4-2 adder compressor ($S = x_1+x_2+x_3+x_4+C_{in}$) is given in Eq. (4.8).

$$S = \text{Sum} + 2(C_{out} + \text{Carry}) \tag{4.8}$$

Figure 4.6 – Internal structure of 4-2 adder compressor



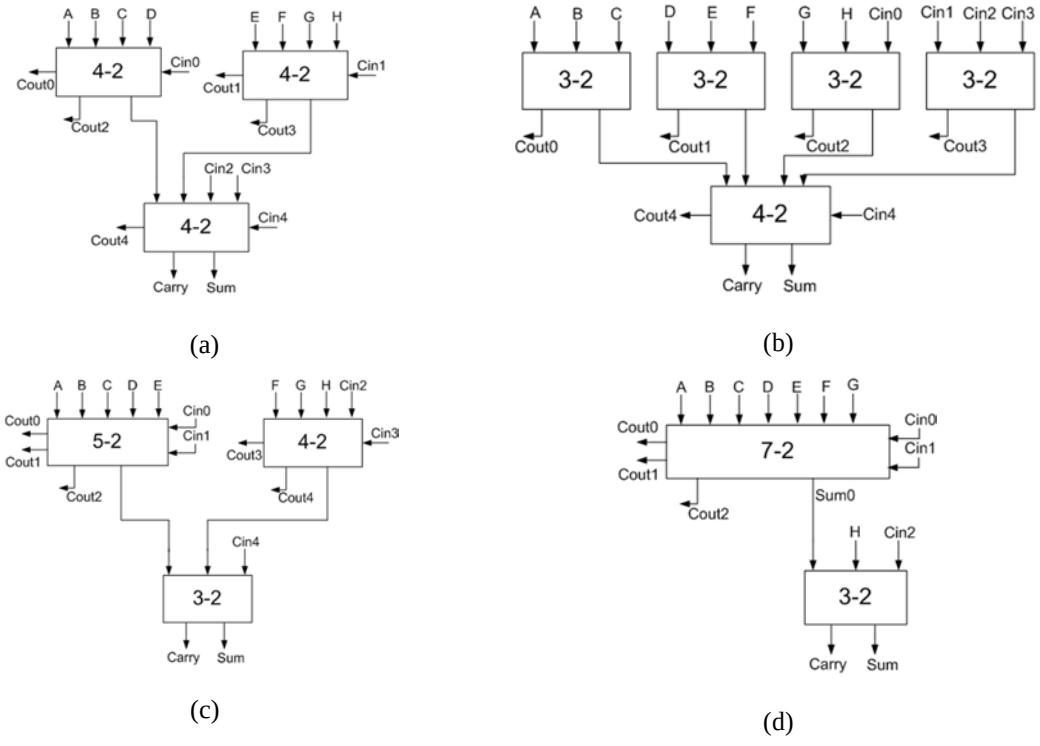
Source: the author, modified from (WEINBERGER, 1981).

The 8-2 adder compressor can be constructed hierarchically using basic 3-2, 4-2, 5-2 and 7-2 adder compressors. We have exploited four different versions of internal structures for the hierarchical 8-2 adder compressor as shown in Figure 4.7. The structures use trees of basic adder compressors.

For the 7-2 adder compressor, we have employed the very efficient structure proposed in (ROUHOLAMINI, 2007), shown in Figure 4.8. In this structure, the critical path is given by 6 EXOR gates. The 7-2 compressor has seven primary inputs, two carry inputs (C_{in}), two carry outputs (C_{out}), and Sum and Carry output terms. All the inputs and output Sum have the same weight. On the other hand, while the output Carry and C_{out1} are weighted one bit order higher, the output C_{out2} has two bit order higher than the inputs. Besides the EXOR gates and MUX, the 7-2 compressor also uses a carry

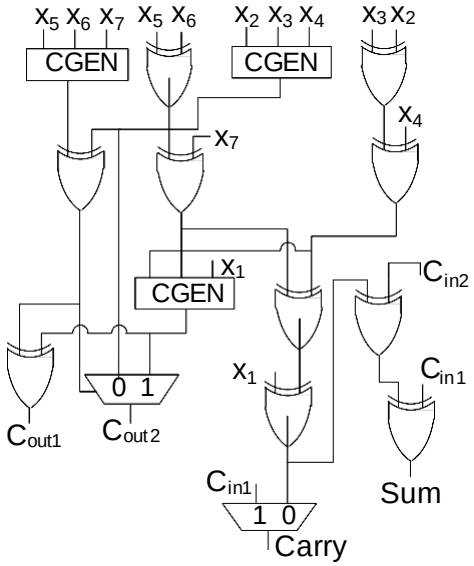
generator module (CGEN). For more details of the 7-2 adder compressor, please refer to (ROUHOLAMINI, 2007).

Figure 4.7 – Hierarchical 8-2 adder compressor using internal structures based on (a) 4-2; (b) 3-2 and 4-2; (c) 5-2, 4-2 and 3-2; (d) 7-2 and 3-2.



Source: (ALTERMANN, 2010).

Figure 4.8 – 7-2 adder compressor structure.



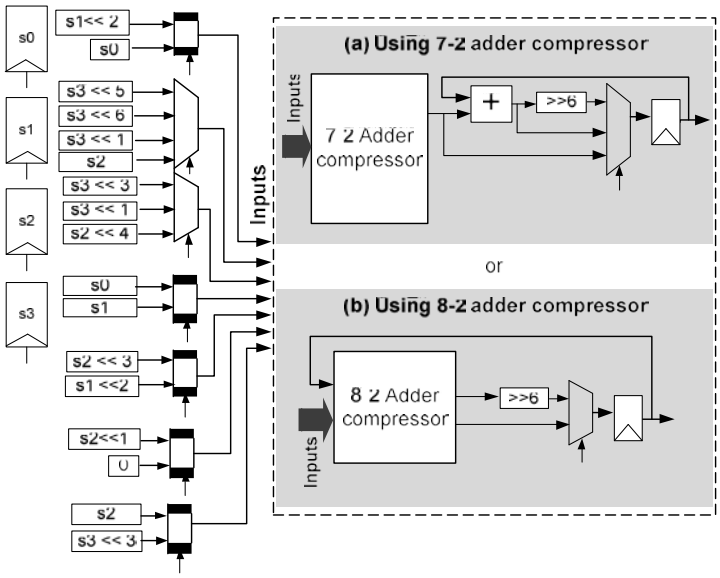
Source: (ROUHOLAMINI, 2007).

4.3.2 Enhancing our Fractional Pixel Interpolation Filter Hardware Architecture with Efficient Adder Compressors

Our Fractional Pixel Interpolation Filter Hardware Architecture for HEVC (Section 4.2) has two acceleration engines, each one composed by 12 interpolation datapaths in parallel. The adder compressors are employed in the interpolation datapaths. The details of internal architecture of the original interpolation datapaths is also shown in Section 4.2. With the observation that luma and chroma interpolation filter datapaths have adder trees, and the intermediate values are not used in the calculation, we can replace the adder trees with efficient adder compressors. In this section, we detail the modified luma and chroma interpolation datapaths by employing adder compressors.

Regarding luma datapath, we have developed two options. The original datapath has 7 values that need to be added, namely the output values from multiplexers. The first option (Figure 4.9a) employs a 7-2 adder compressor (see Section 4.3.1). The second option (Figure 4.9b) includes the accumulator inside the adder compressor, thus employing an 8-2 adder compressor. In both options, the subtraction operator is implemented inside the adder compressors by complementing the values of the negative part of the operation (using the four multiplexers at the bottom of Figure 4.9).

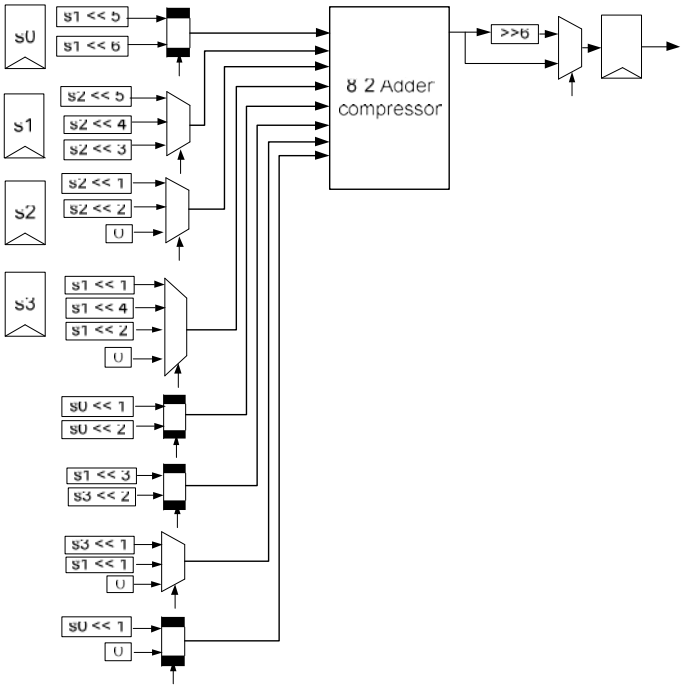
Figure 4.9 – Modified luma filter datapath using (a) 7-2 adder compressor; (b) 8-2 adder compressor.



Source: the author.

Regarding the chroma datapath, it has 8 values to be added (output from multiplexers). We have developed a modified chroma interpolation filter datapath employing an 8-2 adder compressor, as shown in Figure 4.10. In this datapath, we also complement the last four inputs due to the subtraction operation. Four different internal structures of the hierarchical 8-2 adder compressor were exploited (see Section 4.3.1).

Figure 4.10 – Modified chroma filter datapath using 8-2 adder compressor.



Source: the author.

4.3.3 Results and Discussion

The luma and chroma interpolation datapaths were implemented in hardware description language and synthesized into the 45 nm CMOS Nangate Open Cell Library (NANGATE, 2015) using the Cadence RTL Compiler (CADENCE, 2014). The cycle time constraint was set to 2 ns for the automated logic synthesis.

Table 4.3 shows the synthesis results of the datapaths implemented with different adder compressor internal structures, herein referred as implementation versions. Luma datapath has five implementation versions, while chroma datapath has four implementation versions, all shown in Table 4.3. These versions using adder compressors were compared to the datapaths of the original interpolation datapaths (Section 4.2) implemented with Ripple-Carry Adders (RCA). The comparison is done in terms of cell usage, area, delay, power and Power Delay Product (PDP). Power results were generated using simulation of 10,000 random test vectors as inputs into the architecture.

The most power-efficient version for both luma and chroma datapaths is the one using the hierarchical 8-2 adder compressor composed with 3-2 and 4-2 adder compressors (see Figure 4.7b). They dissipate 15% less power than the original architecture with RCA. In terms of area, the most efficient version for luma and chroma datapaths is the one using hierarchical 8-2 compressor composed of 7-2 and 3-2 adder compressors (see Figure 4.7d). They achieve 18% and 13% area reduction in luma and chroma datapaths, respectively, compared with RCA versions. In terms of PDP, the most efficient version for luma and chroma datapaths is the one using hierarchical 8-2 compressor composed by 4-2 adder compressors. They achieve 30% and 18% PDP reduction in luma and chroma datapaths, respectively, compared to RCA versions. The

different area, power and PDP results of the adder compressor implementation versions leave the option to the designer, depending on what is the priority design goal the designer has for the dedicated architecture.

Table 4.3 – Synthesis results for the Interpolation Datapaths.

	Figures	Cells	Area (μm^2)	Delay (ns)	Leakage Power (µW)	Dynamic Power (µW)	Total Power (µW)	PDP ($\times 10^{-13}$)	PDP reduction (%)
Luma datapath									
Original datapath with RCA*	4.3	384	1,692	1.872	26.0	415.9	441.9	8.27	-
i) 7-2	4.9a	337	1,443	1.570	24.4	370.8	395.2	6.20	25
ii) 8-2 (w/ 4-2)	4.9b & 4.7a	392	1,589	1.478	25.6	367.7	393.3	5.81	30
iii) 8-2 (w/ 3-2 and 4-2)	4.9b & 4.7b	395	2,446	1.604	25.2	353.1	378.3	6.06	27
iv) 8-2 (w/ 5-2, 4-2 and 3-2)	4.9b & 4.7c	423	1,702	1.628	25.8	388.6	414.4	6.74	18
v) 8-2 (w/ 7-2 and 3-2)	4.9b & 4.7d	329	1,404	1.549	24.1	365.5	389.6	6.03	27
Chroma datapath									
Original datapath with RCA*	4.4	435	1,761	1.612	27.2	396.8	424.0	6.83	-
i) 8-2 (w/ 4-2)	4.10 & 4.7a	430	1,643	1.466	26.0	355.4	381.5	5.59	18
ii) 8-2 (w/ 3-2 and 4-2)	4.10 & 4.7b	445	1,660	1.563	25.6	337.6	363.2	5.68	17
iii) 8-2 (w/ 5-2, 4-2 and 3-2)	4.10 & 4.7c	449	1,721	1.598	26.1	360.3	386.5	6.18	10
iv) 8-2 (w/ 7-2 and 3-2)	4.10 & 4.7d	396	1,546	1.499	24.7	354.1	378.9	5.68	17

*Uses the datapaths of the architecture in Section 4.2, but implements the adders with Ripple-Carry Adder (RCA).

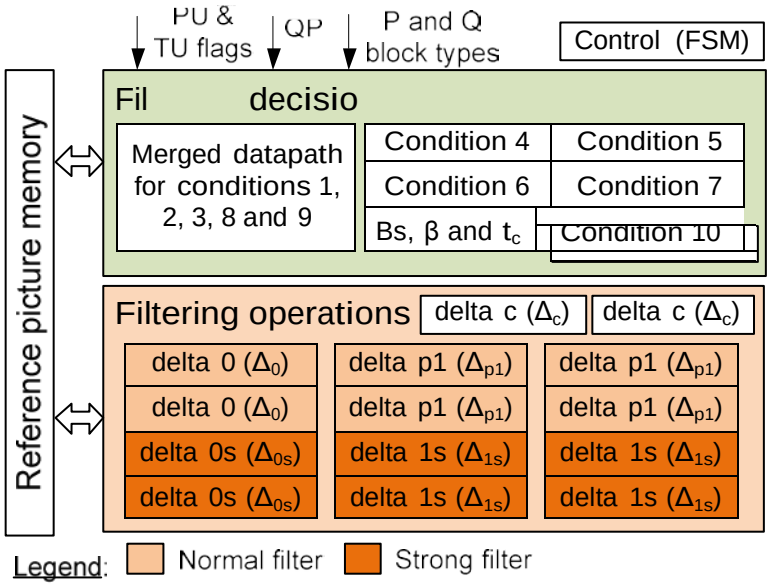
Source: the author.

In the original architecture (Section 4.2), luma and chroma acceleration engines employ 12 datapaths each one. Considering that, we estimate the impact of power in the entire architecture. Employing the most power-efficient implementations versions of luma and chroma datapaths (iii and ii), the architecture dissipates approximately 8.9 mW and spends 49,272 μm^2 of chip area. Related works on HEVC interpolation architectures, (GUO, 2012) and (AFONSO, 2013), do not provide area and power results, so the direct area and power comparisons against their designs are not possible.

4.4 Hardware Architecture for Deblocking Filter of HEVC

This section describes the proposed hardware architecture for Deblocking Filter of HEVC. The system diagram of the architecture is depicted in Figure 4.11. Our architecture receives as input the samples of the 2 neighboring 4x4 blocks (P and Q, please refer to deblocking filter overview in Section 2.3.2). P and Q blocks belong to different adjacent 8x8 blocks in PU or TU boundary to be filtered, which is determined by PU and TU flags. Our architecture decides whether filtering is required or not and the strength of the filtering to be applied if this is the case. It depends on input samples and also on Bs, β and t_c calculated over sample values, P and Q block types, and QP, which are also given as input to our architecture. Our architecture also has some simple control signals to establish a handshake between a master device (e.g. a CPU that runs the whole HEVC encoder/decoder application) and our architecture (which plays the slave role).

Figure 4.11 – System diagram of the proposed hardware architecture for HEVC deblocking filter



Source: (DINIZ, 2015b).

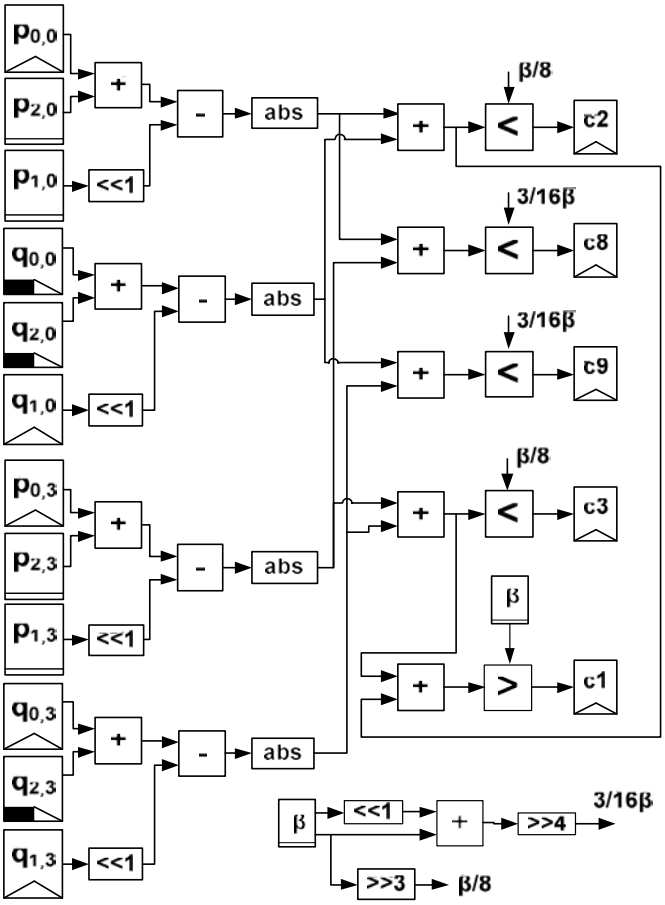
Samples from P and Q blocks to be filtered are first stored into a reference picture memory. As each input sample has 8 bits, our architecture supports a 256-bit wide input memory channel. Therefore, 32 samples (both 4x4 blocks, P and Q) are transmitted in one clock cycle. Filtered samples are stored back into reference picture memory after the filtering operations.

Our architecture has three main units: (1) filtering decisions; (2) filtering operations; and (3) control unit. The filtering decisions unit calculates the conditions to decide whether the boundary must be filtered or not and the strength of the filter. It is composed by some datapaths in parallel, whose internal architecture is detailed in section 4.4.1. If filtering is required, the filtering operations unit calculates the filtered samples. Datapaths of filtering operations are detailed in section 4.4.2. The control unit implements the control flow shown in section 2.3.2 and it establishes the handshake with a master device. The control unit is discussed in section 4.4.3.

4.4.1 Filtering Decisions Datapaths

This unit determines the need of filtering two given 4x4 blocks. For input samples convention and deblocking filter equations, please refer to section 2.3.2. By examining the standard deblocking filtering equations, it can be noted that conditions 1, 2, 3, 8 and 9 share similar sub-expressions with the same input samples. Partial results from conditions 2 and 3 and used for conditions 1, 8 and 9. Hence, we employ hardware reuse to design a merged datapath for those conditions. The diagram of the merged datapath is depicted in Figure 4.12. Conditions 1, 2, 3, 8 and 9 were shortened as c1, c2, c3, c8 and c9, respectively.

Figure 4.12 – Merged datapath for conditions 1, 2, 3, 8 and 9

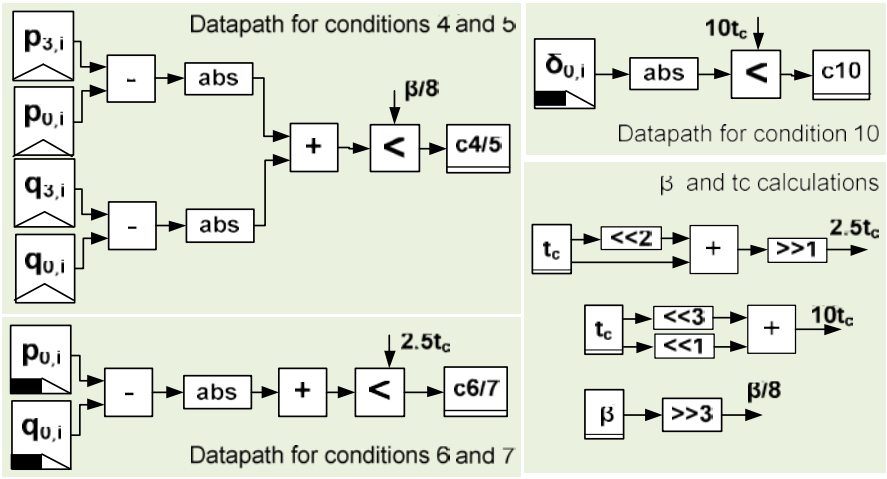


Source: (DINIZ, 2015b).

The condition equations require some multiplication by constants. We have replaced the multiplications by adders and shift operations to use less hardware resources. The proposed merged datapath generates the five conditions in only one clock cycle.

Datapaths for the remaining conditions (4, 5, 6, 7, and 10) are depicted in Figure 4.13. Datapaths for conditions 4 and 5 are equal, only differing by the input samples (condition 4 is applied in the first row and condition 5 is applied in the last row of 4x4 blocks). We have included two instances of this datapath to compute both conditions (c4 and c5) in the same clock cycle. The same was made for conditions 6 and 7. Condition 10 is an additional filtering decision applied to δ_0 for the four rows of 4x4 blocks after normal filtering operation (see more details of filtering operations in section 4.4.2). Our architecture includes two instances of this datapath in the design to compute c10 for the four rows in two clock cycles. Each instance computes two rows of samples. Additional datapaths for β and t_c multiplications, needed to compute all the conditions, are also shown in Figure 4.13. β and t_c values are generated by a lookup table with QP value as input index.

Figure 4.13 – Datapaths for conditions 4, 5, 6, 7 and 10



Source: (DINIZ, 2015b).

4.4.2 Filtering Operations Datapaths

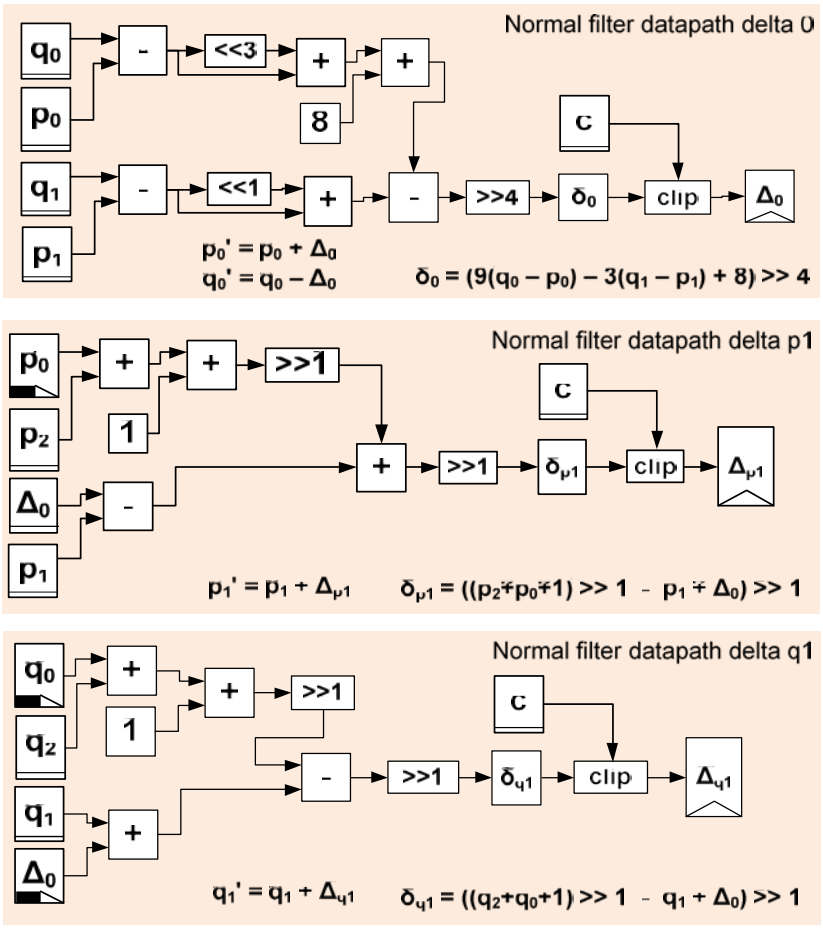
After computing the filtering decisions, this unit computes the filtering operations for normal and strong filters (for luma) and chroma filter.

The clipping operation, included in all the datapaths, is present in Deblocking filter (DF) to prevent excessive blurriness. It keeps the final result inside a range that depends on the QP value, block type and filtering strength. The value c is calculated in the first cycle with the decision process and it is stored in register to be used by the filtering operation datapaths.

Datapaths for normal filtering operations are depicted in Figure 4.14. Normal filter modifies 1 or 2 samples along the block boundary. It computes the delta values (i.e. Δ_0, Δ_{p1}, Δ_{p2}) which are offsets that must be added to the original samples (p_0, p_1, q_0 and q_1) in order to generate the final filtered sample (p_0', p_1', q_0' and q_1'). They are applied to the four rows of samples of 4x4 blocks P and Q. Delta 0 operation is always computed when the normal filter is selected in filtering decision process and modifies the p_0 and q_0 samples that are close to the boundary. Delta p1 and delta q1 operations modify also p_1 and q_1 samples. Please refer to deblocking filter flow in section 2.3.2 for details. In our architecture, delta p1 and delta q1 values are computed anyway in the same cycle to achieve high throughput and to simplify control. However, the final sample modification depends upon the result of conditions 8 and 9. This is done by the control unit.

In the design of this unit we have also replaced the multiplications by constant by a sequence of adders and shift operations. Our architectural design includes two instances of each datapath shown in Figure 4.14 (please refer to the architecture diagram in Figure 4.6). This way, we can compute the normal filtering operations in two cycles. Each datapath instance computes two rows of samples.

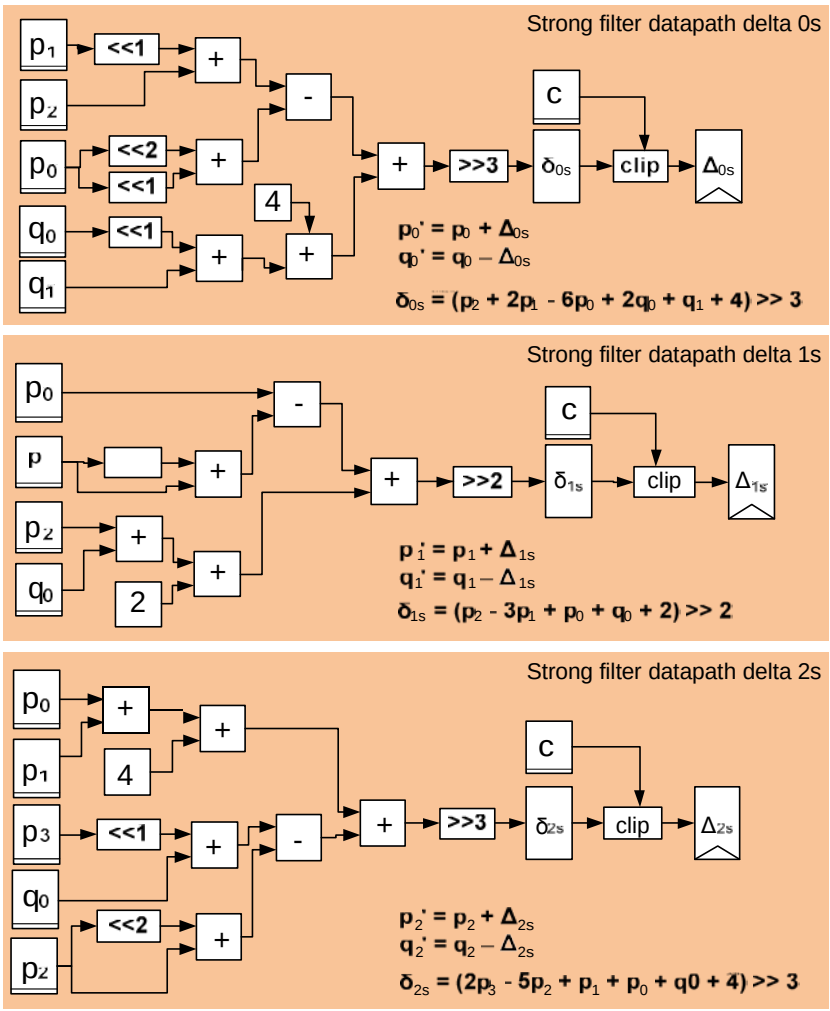
Figure 4.14 – Datapaths for normal filtering operations



Source: (DINIZ, 2015b).

Strong filtering datapaths are depicted in Figure 4.15. Strong filter always modify 3 samples along the block boundary. It computes the delta values (i.e. Δ_{0s}, Δ_{1s}, Δ_{2s}) which are offsets that must be added to the original samples (p_0, p_1, p_2, q_0, q_1 and q_2) in order to generate the final filtered sample (p_0', p_1', p_2', q_0', q_1' and q_2'), as shown in Figure 4.15. They are also applied to the four rows of samples of 4x4 blocks P and Q. Similar to normal filter datapaths, we have also replaced the multiplications by adders and shift operations and we have included two instances of each datapath in the architecture. Each datapath instance computes two rows of samples.

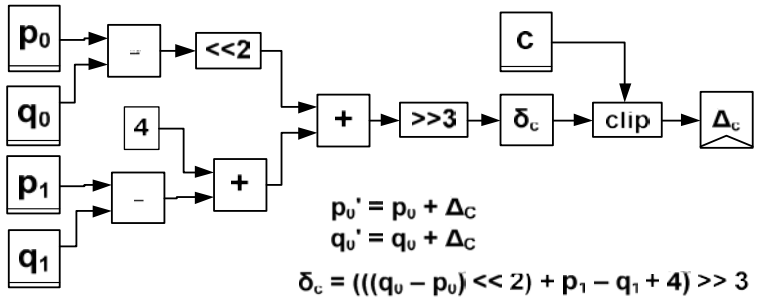
Figure 4.15 – Datapaths for strong filtering operations



Source: (DINIZ, 2015b).

Our architecture also includes two instances of the datapath for the chroma filtering. The diagram of chroma filter datapath is shown in Figure 4.16. Chroma filter is only applied when Bs is equal to 2 and does not require further decisions. It modifies only p_0 and q_0 samples.

Figure 4.16 – Datapath for chroma filtering operation



Source: (DINIZ, 2015b).

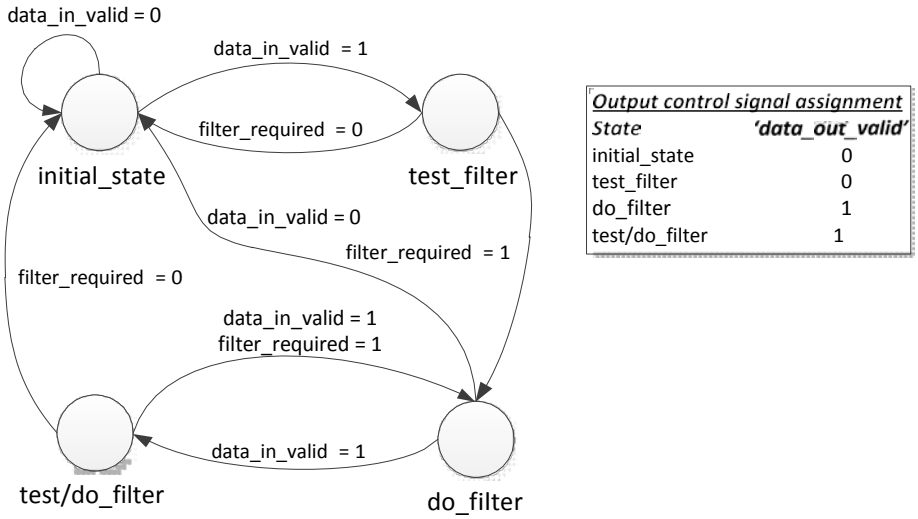
After generating the delta values, they are added with the original samples to produce the final filtered samples, depending on the results of conditions calculated in the filtering decisions unit. One multiplexer is included for each output.

4.4.3 Control Unit

Due to the distinct dataflow nature of the deblocking filter, the control unit of the architecture is relatively simple. A Finite State Machine (FSM) handles the handshake protocol between master and slave and selects the correct filtered output samples based on the filtering conditions unit. The FSM has 4 states, as shown in Figure 4.17. The output values are also described in Figure 4.17.

The master starts a transmission by signalizing with ‘data_in_valid’ signal. Then, the deblocking filter architecture reads the input data port from memory. At the next clock cycle, if filtering of the given input is required, the filtered samples are available at the output data port and ‘data_out_valid’ signal changes to ‘1’. The deblocking filter architecture expects to receive the next 2 rows of samples to be filtered. At the next clock cycle, the filtered samples are available at the output data port. This process can be repeated, resulting in new blocks at every 2 clock cycles or no filtered block (when filtering is not necessary) at every clock cycle. The master can stop transmitting data at the end of the second clock cycle, by signalizing ‘data_in_valid’ with ‘0’.

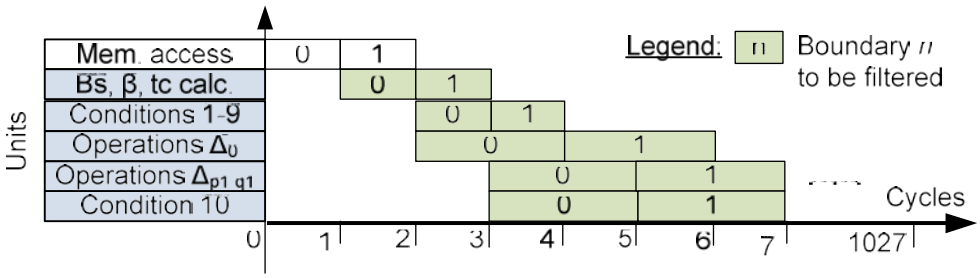
Figure 4.17 – State diagram of the Finite State Machine



Source: (DINIZ, 2015b).

An example of the processing scheduling of the deblocking filter architecture is shown in Figure 4.18. We show in this example the normal filter operation which needs to modify two samples along the boundary. This is the worst case condition for our architecture, because it needs more clock cycles to complete. Other situations, e.g. no filtering, strong filtering, or even normal filtering modifying only one sample along the boundary require less clock cycles to complete the calculation (please refer to the deblocking filter flow in section 2.3.2 and the control state machine in Figure 4.17). Figure 4.18 shows the scheduling of processing block boundaries of 4 samples to be filtered (P and Q blocks). The numbers inside the boxes are the boundaries’ numbers.

Figure 4.18 – Processing schedule of normal filter (worst case)



Source: (DINIZ, 2015b).

Our architecture has an initial latency of 5 clock cycles to complete normal filter (modifying 2 samples along the boundary) for the first four-sample boundary. After the initial latency, it delivers a new normal filtered boundary at each 2 clock cycles, due to our pipelined architecture. We have considered a worst-case condition for our architecture as follows: when a CTU (64x64-pixel block) of video is all partitioned into 8x8-pixel blocks and all boundaries between those blocks need to be filtered by the normal filtering that modifies 2 samples along the boundary. This is a very worst case condition, since not always the CTU is all partitioned into 8x8-pixel blocks and not always it need to be filtered by the worst case filtering mode. However, this analysis is useful to compare the architecture with state of the art. Hence, considering the worst case condition, our architecture requires 1,027 clock cycles to filter one CTU.

4.4.4 Results and Evaluation

The proposed deblocking filter hardware architecture was implemented in VHDL and synthesized to ASIC and FPGA. FPGA synthesis and mapping is performed using Xilinx ISE design suite (XILINX, 2014) for Xilinx Virtex-6 XC6VLX130T-ff1156-3 FPGA device (XILINX, 2012). ASIC synthesis is performed using Cadence RTL Compiler tool (CADENCE, 2014) for FreePDK 45nm CMOS standard-cell library (FREEPDK, 2014) under a constraint of 200 MHz clock frequency. Area results for the target clock frequency for both FPGA and ASIC are shown in Table 4.4. Area results for FPGA are shown in terms of Slice Look-up Tables (LUTs) and slice registers. ASIC area results are shown in terms of gate count (NAND-2 area equivalent) to enable comparisons with related works which are implemented in different CMOS technologies. It can be noted that our hardware architecture requires very low hardware resources due to our architecture-independent and hardware-specific optimizations to create very optimized datapaths for HEVC deblocking filter.

Table 4.4 – Synthesis results of the deblocking filter architecture for FPGA and ASIC

Hardware units	FPGA results		ASIC results
	Slice LUTs	Slice Registers	Gate count (NAND2)
Conditions	383	168	980
Operations	770	265	1,931
Control (FSM)	245	8	370
Total	1,398	441	3,281
Frequency (MHz)	140		200

Source: (DINIZ, 2015b).

Table 4.5 shows the comparison of our architecture against two state of the art references (OZCAN, 2013) (SHEN, 2013) which implement in hardware the deblocking filter. The work in (SHEN, 2013a) presents a DF architecture which is very similar to the work in (SHEN, 2013). Herein only the work in (SHEN, 2013) is being considered for comparison purpose.

Table 4.5 – Comparisons to the state of the art hardware implementations of the deblocking filter

	(OZCAN, 2013)	(SHEN, 2013)	Our architecture
Architecture parameters			
Cycles/CTU (worst case)	7,680	440	1,027
Minimum frequency (MHz) to process 1920x1080@30fps	108	7	15.6
Minimum frequency (MHz) to process 4096x2048@60fps	–	56	124.8
ASIC implementation			
CMOS technology (nm)	90	130	45
Maximum frequency (MHz)	108	200	200
Gate count (NAND2 equiv.)	16.4k	21k	3.3k
Maximum throughput (resolution @ frame rate)	1920x1080@ 30 fps	4096x2048 @ 60 fps	4096x2048 @ 60 fps
Power (mW)	–	–	4.6
FPGA implementation			
Device	Virtex-6	–	Virtex-6
Frequency (MHz)	108	–	140
Slice LUTs	5,236	–	1,398
Slice registers	1,547	–	441
Maximum throughput (resolution @ frame rate)	1920x1080@ 30 fps	–	4096x2048 @ 60 fps
Power (mW)	31.0	–	9.0

Source: the author.

Compared to the work in (OZCAN, 2013), the ASIC implementation of our architecture reduces gate count by approximately 5X, while providing 4X maximum throughput in terms of real-time video processing for certain video resolution and frame rate. The FPGA implementation reduces LUTs and slice registers by approximately 4X, while providing high throughput with a maximum frequency of 140 MHz. We have synthesized our design for the same FPGA device of (OZCAN, 2013) to enable direct comparison in terms of slice LUTs and slice registers being used in the configuration. The throughput of our architecture is higher than the related work mainly due to the reduced number of cycles/CTU (in the worst case) of our architecture. Our architecture reduces the cycles/CTU by more than 7X. Area is reduced in our design through our architecture independent and hardware specific optimizations, i.e. reusing of operations, replacing multiplications by add/shift operations.

The work in (SHEN, 2013) has a more efficient design than (OZCAN, 2013) at the cost of a larger hardware area. It does not include FPGA implementation, so comparisons are possible only against our ASIC results. Compared to (SHEN, 2013) the ASIC implementation of our architecture reduces gate count by more than 6X, while providing a similar throughput. Another comparison is conducted in terms of the architecture (and implementation-independent) parameters: cycles/CTU (worst case) and minimum frequency to process two video resolutions in real time: 1920x1080 @ 30

fps; and 4096x2048 @ 60 fps. Our work requires twice the minimum frequency to process videos at a certain frame rate (15.6 MHz vs. 7 MHz, and 124.8 MHz vs. 56 MHz) than (SHEN, 2013). However, our architecture achieves 6X gate count reduction compared to (SHEN, 2013). By increasing operating frequency to only 124.8 MHz, our design achieves throughput to process 4096x2048@60 fps. Both FPGA and ASIC implementations of our architecture reach this operating frequency, as shown in the results in Table 4.4.

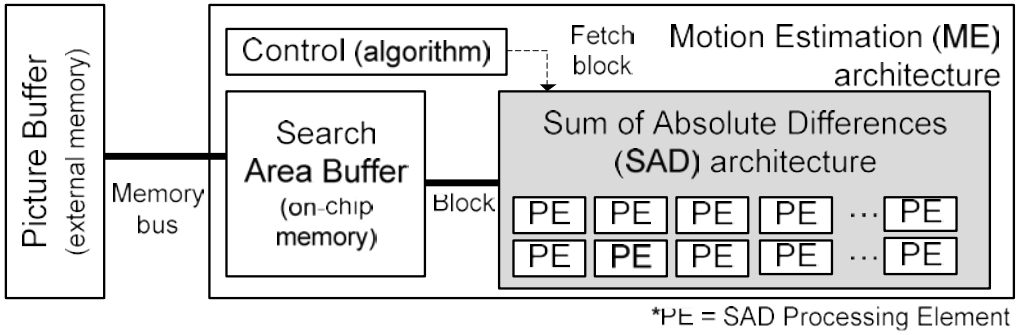
We have also provided power estimation of our architecture. Regarding the ASIC implementation, it was not possible to compare power estimation results since the authors of (SHEN, 2013) and (OZCAN, 2013) do not provide any power estimation numbers for ASIC implementations. (OZCAN, 2013) quotes its power estimation for FPGA implementation. We estimated power of our architecture and the architecture proposed in (OZCAN, 2013), both implemented in FPGA, with the proprietary Xilinx power estimator tool (XILINX, 2013). Our architecture dissipates 3X less power compared to the architecture proposed by (OZCAN, 2013).

4.5 Hardware Architecture for Sum of Absolute Differences (SAD)

As reviewed in chapter 2, SAD is a low-complexity distortion metric used in mode decision and Motion Estimation (ME) stages of advanced video encoders. Considering our HEVC application analysis in chapter 3, it consumes a huge amount of total encoding execution time. It is definitely an important computational kernel in HEVC encoder. Many related works proposed different hardware architectures for SAD calculation. Most of them are included in complete ME hardware architectures targeting H.264/AVC and HEVC standards.

An abstract system diagram of a ME hardware architecture is shown in Figure 4.19. It usually includes three modules: i) on-chip memory to store search area buffer; ii) control unit that implements the ME algorithm; and iii) a SAD architecture with many SAD processing elements (PEs) in parallel. Search area buffer stores the candidate blocks to be compared with the current block to be encoded. This buffer is filled in advance by fetching the search area from the picture buffer on external memory. The control unit implements the ME algorithm (usually a fast block matching algorithm) and delivers the candidate blocks to be compared with the current block by the SAD architecture. The SAD architecture contains many PEs in parallel because the candidates can be compared in parallel with the current block since there is no dependency among different block comparison.

Figure 4.19 – A Motion Estimation (ME) architecture diagram and the Sum of Absolute Differences (SAD) architecture



Source: the author.

In this thesis there is a design space exploration for the architecture of the SAD processing element design. We have designed nine hardware architecture alternatives for SAD processing element, varying the parallelism level (4, 8 and 16 samples in parallel) and the number of pipeline stages. Then, we conduct a comparative analysis of the architectural alternatives of SAD PE in terms of hardware area, throughput, power, and energy consumption. Depending on system requirements, the designer may choose a highly-parallel deep-pipelined SAD version, e.g. to achieve high performance, or a less parallel SAD version to dissipate lower power and consume lower energy. Other modules of a complete ME architecture, e.g. the ME algorithm, control, and memory fetching, are not the focus of this thesis.

4.5.1 Exploiting Different Versions of Parallel SAD Processing Elements

We designed different hardware architecture alternatives for SAD processing elements following a fixed structure. The generic structure of a SAD PE is defined as follows, with N being the number of 8-bit input samples in parallel:

- N 8-bit subtractors in parallel to subtract N original samples (of current block) from N predicted samples (of a candidate block);
- N absolute (abs) operations in parallel;
- A set of adders organized in a binary-tree, with N input values and $\log_2 N$ deep (uses N-1 adders);
- An accumulator (an adder and a register) to add partial SAD values to compose the final SAD.

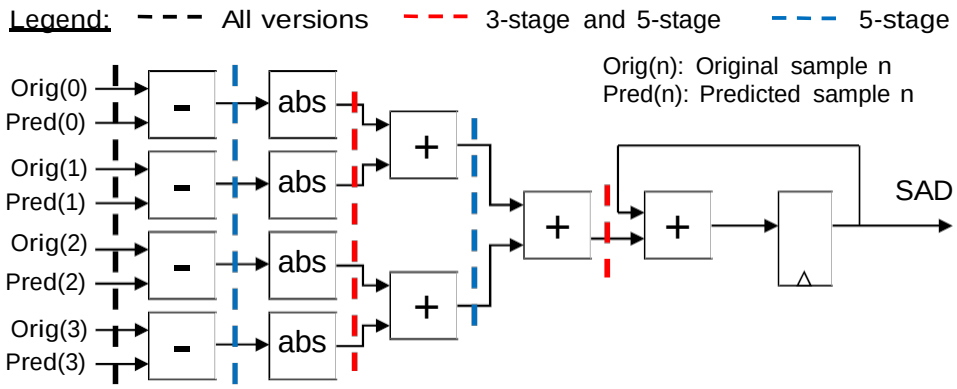
We have designed nine architecture alternatives for SAD PE with a combination of two parameters: i) the number of input samples in parallel (4, 8 and 16); ii) the number of pipeline stages, depending on each version of parallelism. Figures 4.20, 4.21 and 4.22 show the SAD PE architecture alternatives with 4-input samples, 8-input samples and 16-input samples, respectively.

The advantage to explore the parallelism is to increase the architecture throughput to perform SAD calculation of more samples per clock cycle. However, doubling the number of input samples increase in one adder the depth of adder tree, i.e. one adder is included in the critical path for pure combinational version (1-stage). To analyze this issue three architecture alternatives were designed for each parallelism version varying the number of pipeline stages, as shown in Figures 4.20, 4.21 and 4.22:

a) 1-stage, 3-stage and 5-stage pipeline alternatives for 4-input samples SAD processing element;
b) 1-stage, 3-stage and 6-stage pipeline alternatives for 8-input samples SAD processing element;
c) 1-stage, 4-stage and 7-stage pipeline alternatives for 16-input samples SAD processing element.

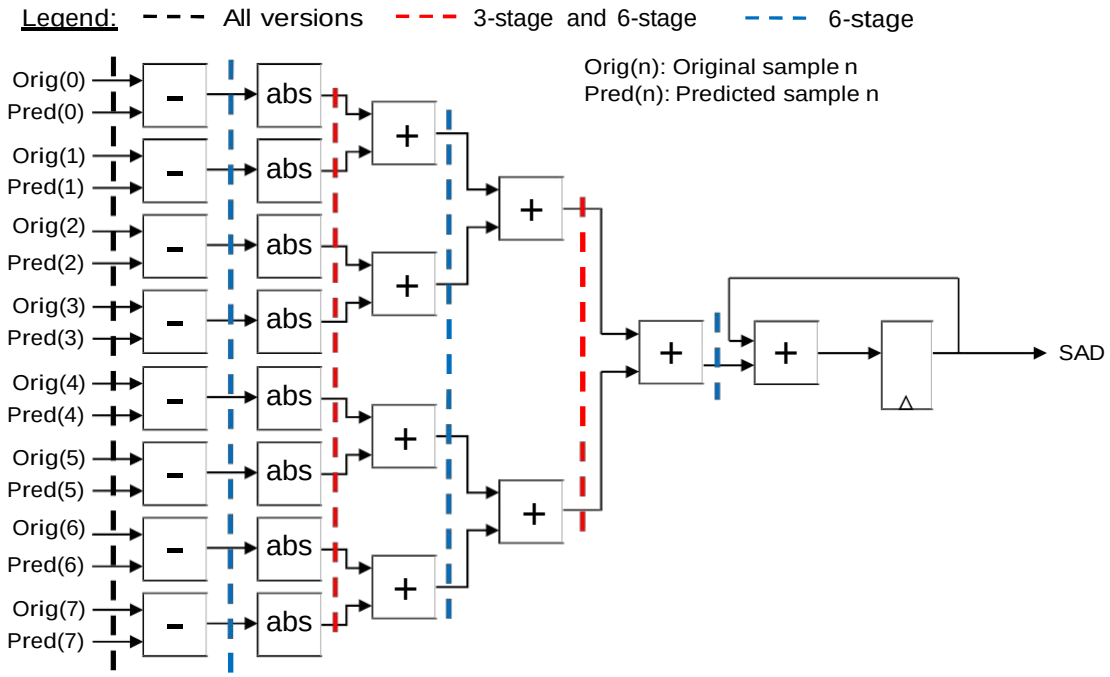
All versions contain registers at the input (Orig and Pred samples) and the output (register to store the final SAD value). The alternatives differ on where the pipeline registers are present, represented by dashed lines in distinct colors. At each stage of the adder tree, the dynamic range of adders is increased by 1-bit.

Figure 4.20 – SAD Processing Element (PE) alternatives with 4-input samples



Source: (DINIZ, 2010).

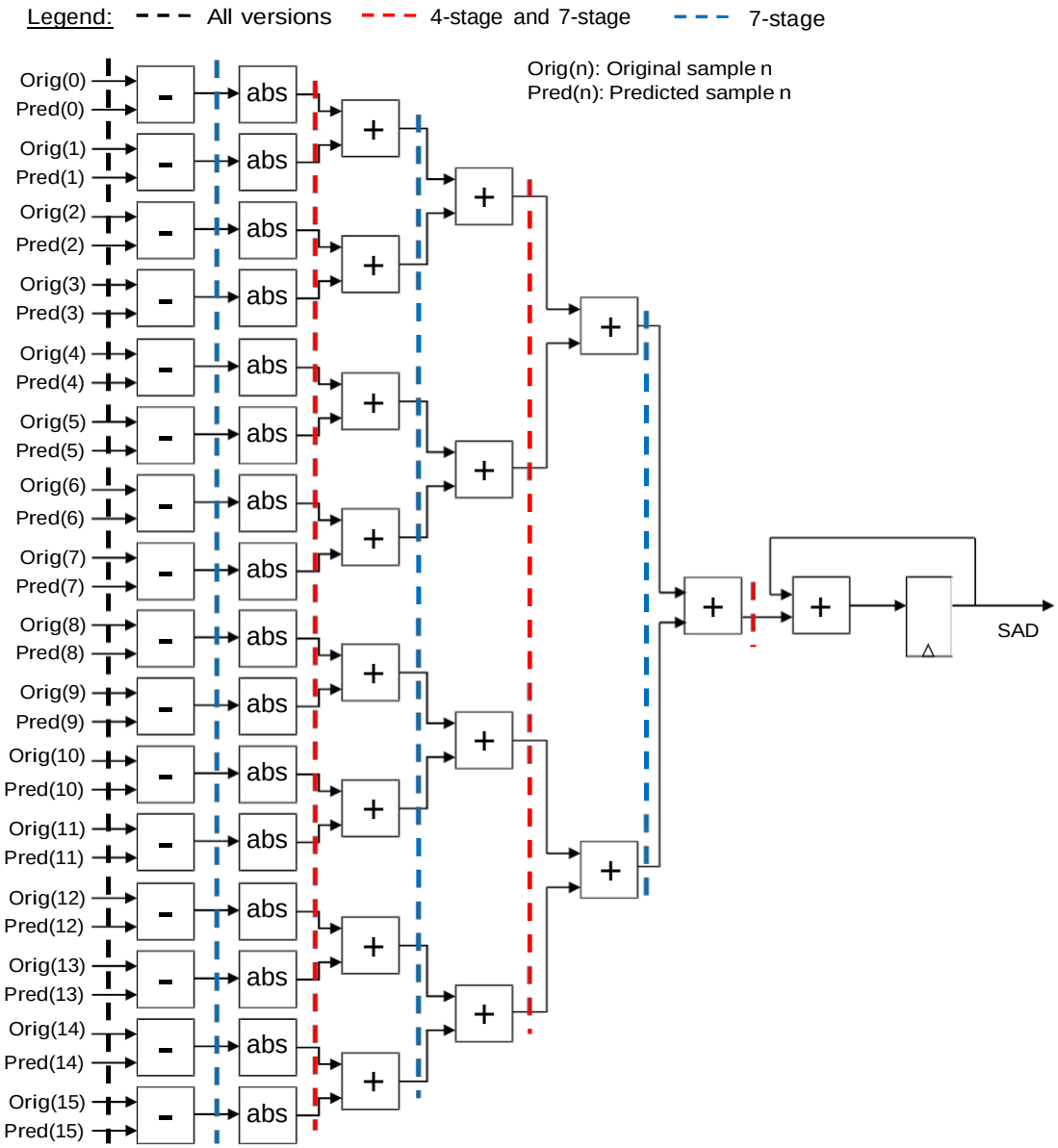
Figure 4.21 – SAD Processing Element (PE) alternatives with 8-input samples



Source: (DINIZ, 2010).

Table 4.6 shows the architectural parameters of the different SAD PE architectural alternatives that are evaluated in this work. Since they have different input samples and pipeline stages, they have different number of clock cycles per block. We have considered a block of size 64×64 samples since this is the larger block size defined in HEVC standard. The number of pipeline stages also defines the initial latency (in clock cycles) of each architectural alternative.

Figure 4.22 – SAD Processing Element (PE) alternatives with 16-input samples



Source: the author.

Table 4.6 – Architectural parameters of the different SAD PE alternatives

SAD PE alternative		Initial latency (cycles)	Cycles/block (one 64x64 candidate block, includes initial latency)
Input samples	Pipeline stages		
4	1	1	1025
4	3	3	1027
4	5	5	1029
8	1	1	513
8	3	3	515
8	6	6	518
16	1	1	257
16	4	4	260
16	7	7	263

Source: the author.

4.5.2 Results and Evaluation

All the SAD PE architectural alternatives were described in VHDL and synthesized to FreePDK 45 nm (FREEPDK, 2014) CMOS standard-cells library using Cadence RTL Compiler tool (CADENCE, 2014). Table 4.7 shows the results and comparisons of the architecture alternatives for SAD PE in terms of area (gate count, 2-input NAND equivalent), maximum frequency, performance, and power.

Table 4.7 – Synthesis results and comparison of the different SAD PE alternatives

SAD PE alternative		Gate count	Max. frequency (MHz)	Performance (64x64 blocks/s)	Power (µW) @ max. freq.
Input samples	Pipeline stages				
4	1	1,097	384	374,634	994
4	3	1,420	714	695,228	1,532
4	5	1,920	769	747,327	2,315
8	1	1,995	370	721,247	1,820
8	3	2,631	667	1,295,145	2,896
8	6	3,726	714	1,378,378	4,617
16	1	4,120	344	1,338,521	3,475
16	4	5,171	526	2,023,076	5,777
16	7	7,300	667	2,536,121	9,230

Source: the author.

The performance is obtained in terms of the number of candidate 64x64 blocks (the larger block size defined in HEVC standard) processed per second with the SAD calculation. We show performance results considering the maximum operating frequency of each alternative obtained after synthesis process. The alternative with 16-input samples and 7-stage pipeline achieves the highest performance, which is 6.7X the performance of the lowest performance architectural alternative, i.e. the 4-input samples 1-stage pipeline alternative, at the cost of 9.28X higher power dissipation.

Hence, for SAD designs in which a power constraint is imposed, alternatives with lower performance, but also lower power dissipation, may be preferred. The designer may choose lower parallelism/pipeline alternatives when power constraint is imposed. Deeper pipeline versions dissipate more power because of the higher dynamic power dissipated on the clock circuit of pipeline registers. Increasing the number of input samples (i.e. increasing parallelism) also results in higher power dissipation, since there is a higher number of operators and consequently more gates operating and dissipating in parallel.

5 RECONFIGURABLE HARDWARE ARCHITECTURE FOR FRACTIONAL-PIXEL INTERPOLATION OF HEVC

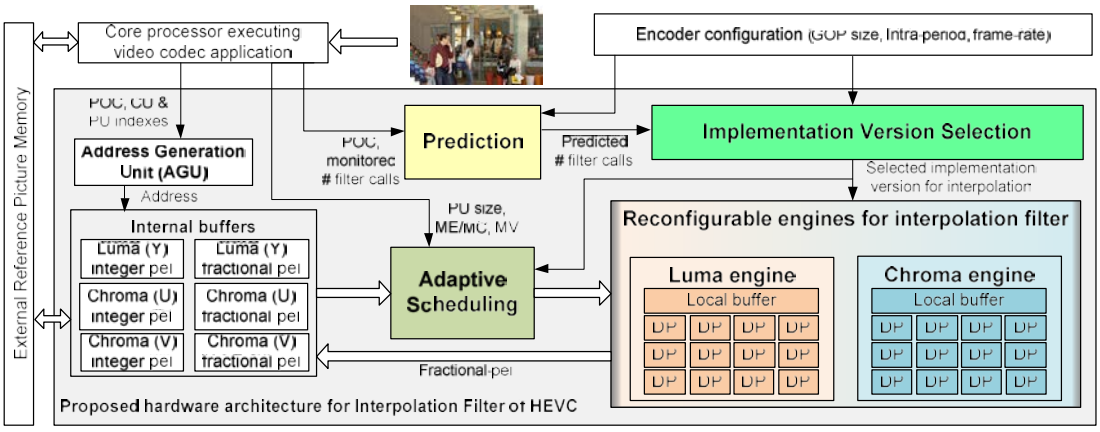
Dedicated hardware architectures provide high performance and energy efficiency for real-time video encoding and decoding, as discussed in Chapter 4. However, dedicated hardware architectures have some drawbacks. First, they are fixed in design time and cannot change the hardware in the field, after silicon fabrication. Second, they incur in high Non-Recurring Engineering (NRE) cost (the one-time cost to design and test a new chip) and high design time. In this scenario, reconfigurable hardware, especially FPGAs, provides a platform solution with low NRE cost, faster time-to-market, and flexibility of quick upgrades through dynamic reconfigurations (TUAN, 2006). FPGA-based designs combine the performance and efficiency of dedicated accelerators due to their capability to exploit high degree of parallelism along with a high degree of flexibility due to their programmability and hardware reconfigurability (SHAFIQUE, 2009)(COMPTON, 2002).

This chapter describes a novel reconfigurable hardware architecture for interpolation filtering in HEVC. Unlike our dedicated interpolation filter architecture discussed in sections 4.2 and 4.3, and different of other state-of-the-art techniques, the architecture presented in this chapter adapts depending upon the coding configurations and throughput requirements through run-time reconfiguration of adaptive datapaths. It thereby provides the performance and power efficiency of dedicated accelerators along with the high flexibility due to run-time reconfigurability. The proposed architecture is beneficial for low-volume productions, where short time-to-market, low NRE, and adaptivity to different coding scenarios are required. Moreover, due to the benefits of partial reconfiguration feature, the proposed architecture is especially beneficial for small-sized FPGAs. The general applicability of our architecture is FPGA-based video encoding systems.

Figure 5.1 shows an overview of our novel reconfigurable hardware architecture for the HEVC interpolation filter. It is composed of four main modules:

1) The Prediction Module (Section 5.1) provides an estimate of the number of interpolation filter calls for the upcoming pictures based on the monitored GOP-history.
2) Reconfigurable Hardware Accelerator Engines for the Luma and Chroma interpolation filters (Section 5.2) with a set of different implementation versions providing multiple area vs. performance/throughput tradeoff options.
3) An Implementation Version Selection Module (Section 5.3) to select an appropriate filter implementation for the reconfigurable engine based on the predicted number of calls.
4) An Adaptive Scheduling Module (Section 5.4) to determine the processing order and the filter type configuration in an adaptive way.

Figure 5.1 – Proposed reconfigurable hardware architecture for Interpolation Filter of HEVC



Source: (DINIZ, 2015a).

The proposed hardware architecture is tightly connected to a core processor (e.g., a Leon-II or NIOS) that executes the video codec application. During the run time, it provides the Picture Order Count (POC), indexes for Coding Unit (CU) and Prediction Unit (PU), PU size, and the monitored number of filter calls for each picture. In case of the HEVC encoder, additional input parameters are: GOP size, Intra-period and the frame-rate. We also assume that the acceleration engines are reconfigured into Partial Run-Time Reconfiguration (PRR) regions as supported by the current FPGA technology (XILINX, 2010) (ALTERA, 2010).

Our architecture employs internal buffers to store integer-pels from external reference picture memory and fractional-pels delivered by the reconfigurable engines. The Address Generation Unit (AGU) translates POC, CU and PU indexes into internal address representation for the internal buffers.

In the following sections, we discuss the main modules in more details. Section 5.5 presents results, evaluation, comparison with related work and discussion.

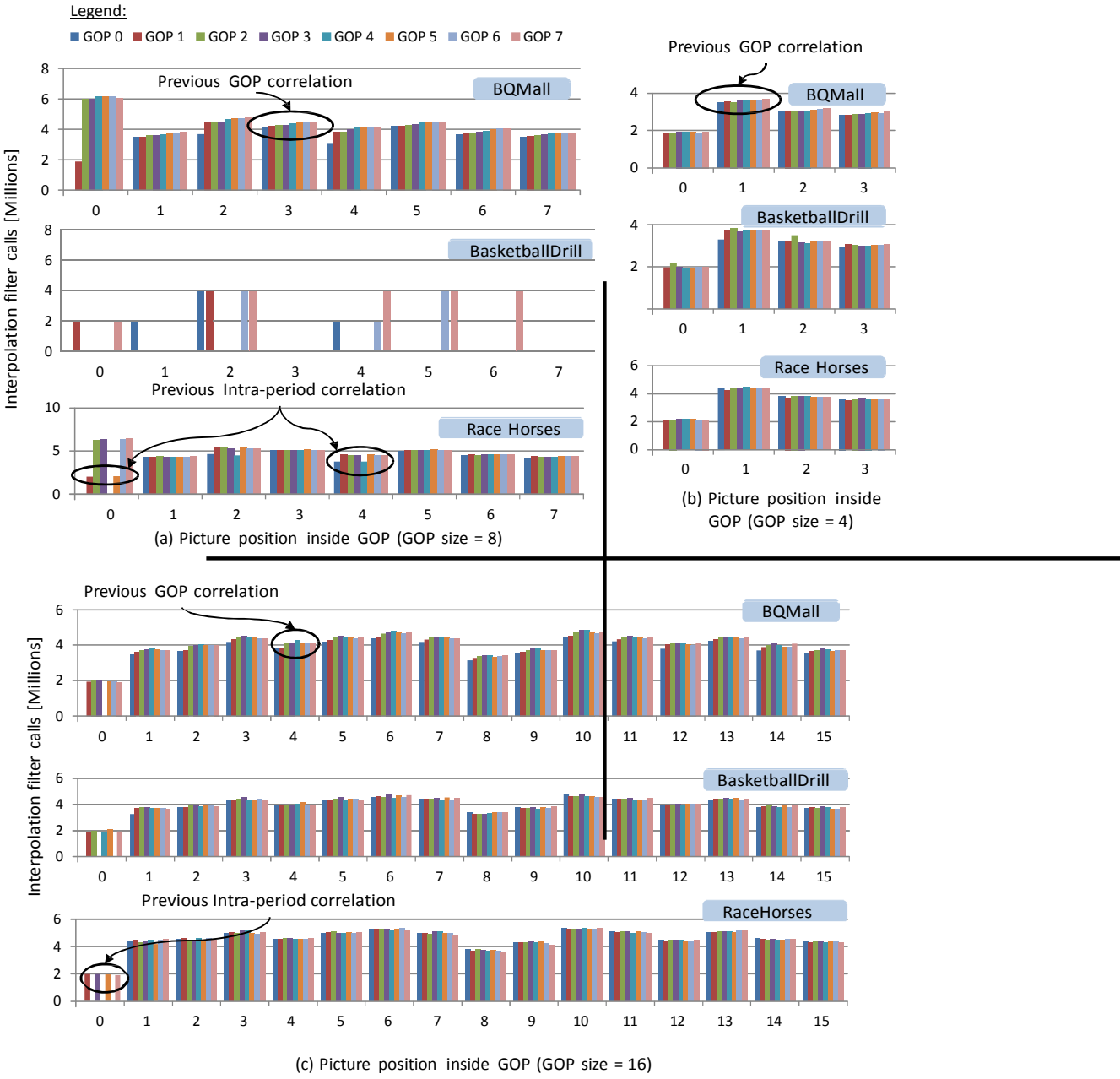
5.1 Adaptive Prediction of Interpolation Filter Calls

5.1.1 Analytical Observations

To design a robust prediction, we need to analyze the pattern of interpolation filter calls within a GOP. Figure 5.2 illustrates the number of interpolation filter calls for the first eight GOPs of three WVGA (832x480 pixels) video sequences BQMall, BasketballDrill, and Race Horses with 60 fps, 50 fps, and 30 fps, respectively. Common test conditions are: RA configuration, QP=22 and GOP sizes={4,8,16}. The numbers of filter calls among pictures at the same relative position inside GOP were compared.

Figure 5.2 shows that the pictures within a GOP have good correlation (in terms of filter calls) with their collocated pictures in the previous GOPs. Therefore, the number of filter calls for the to-be-soon-encoded pictures within a GOP can be accurately predicted using the monitored filter calls in their collocated pictures in the previous GOPs.

Figure 5.2 – Correlation of the number of interpolation filter calls considering GOP sizes equal to (a) 8, (b) 16, and (c) 4



Source: (DINIZ, 2015a).

While most of pictures exhibit good correlation with pictures in previous GOP, there are some exceptions for the pictures 2 and 4 of the GOP 0 and picture 0 of GOP 1. They exhibit fewer number of filter calls than those collocated pictures in the next GOPs because they use Intra picture as a reference for prediction. Therefore, in these cases, the result of Integer Motion Estimation is improved and requires less effort for the Fractional Motion Estimation leading to a reduced number of filter calls. In such cases, we observe that these pictures exhibit a better correlation with the collocated pictures in the previous Intra-period compared to the collocated pictures in the previous GOP. This behavior is due to the hierarchical bi-predictive coding structure used in HEVC (KIM,

2013). In this structure, pictures between two successive Intra pictures are encoded as B pictures. The Intra picture is encoded using a given QP value. The first B picture of each GOP (except GOP 0) is called Generalized P and B (GPB) picture. It uses 4 pictures as reference and is encoded using QP+1 value. Other pictures are divided in three layers and are encoded using QP+2, QP+3 and QP+4 values (VANNE, 2012) (KIM, 2013). The number of interpolation calls increases when more pictures are used as reference (due to bi-predictive ME) and when QP is low (higher quality require more efficient prediction). For that reason, the number of interpolation filter calls is higher in GPB picture than others (see picture 0 of Figure 5.2a). For some GPB pictures, the lower number of filter calls is because they use Intra picture as reference.

In summary, there are two different types of correlations for the number of filter calls: (1) correlation with the collocated in previously encoded GOPs; (2) correlation with the collocated in previous Intra periods.

5.1.2 Prediction Design

Based on the above analytical observations, we designed two types of predictions for the number of filter calls. The first one calculates a Predicted Number of Filter Calls (PNFC) for a current picture with position p[0..GOP_size] inside GOP of index g, based on the Monitored Number of Filter Calls (MNFC) of collocated picture in position p from the previous GOP (see Equation 5.1). The position p of a picture inside GOP is calculated from Picture Order Count (POC) and GOP size, i.e., p = POC mod GOP_size. In order to improve the prediction quality, the prediction error ε (Equation 5.2) for the previous GOP (i.e., ε(p,g-1)) is added in a weighted way, such that the weighting factor is given as δ_1.

$$PNFC(p,g) = MNFC(p,g-1) + \delta_1 * \varepsilon(p,g-1) \quad (5.1)$$

$$\varepsilon(p,g) = MNFC(p,g) - PNFC(p,g) \quad (5.2)$$

The second type of prediction shown in Equation (5.3) computes a PNFC for the current picture p and GOP g based on MNFC of the collocated picture in the previous Intra-period g_{PREV_IP}, as shown in Equation (5.4). In order to improve the prediction quality, the prediction error ε (Eq. 2) for the previous Intra-period (i.e., $\varepsilon(p,g_{PREV\_IP})$) is added in a weighted way, such that the weighting factor is given as δ_2. The division in Equation (5.4) is an integer division as the Intra-period is always in multiples of the GOP size.

$$PNFC(p,g) = MNFC(p,g_{PREV\_IP}) + \delta_2 * \varepsilon(p,g_{PREV\_IP}) \quad (5.3)$$

$$g_{PREV\_IP} = g - Intra\_period / GOP\_size \quad (5.4)$$

Based on the observations in Figure 5.2a and Figure 5.2c, we apply the second type of prediction, as shown in Equation (5.3) for the GPB picture (position 0) and for the picture in the center of the GOP (position 4). These pictures have a lower QP and a higher number of reference pictures, so they exhibit a high number of interpolation filter calls. A different behavior happens with the first picture in position 2 inside an Intra-period. It exhibits a considerably less number of filter calls than the average. This is because this picture uses Intra picture as reference, so the IME finds a good match without FME refinement. Equation (5.3) is also used for picture in position 2 in the first GOP. The following pictures in position 2 have a higher number of filter calls than the first one, but a higher correlation with each other. Therefore, Equation (5.3) is also used for the picture in position 10 inside an Intra-period (i.e. picture position 2 of the second

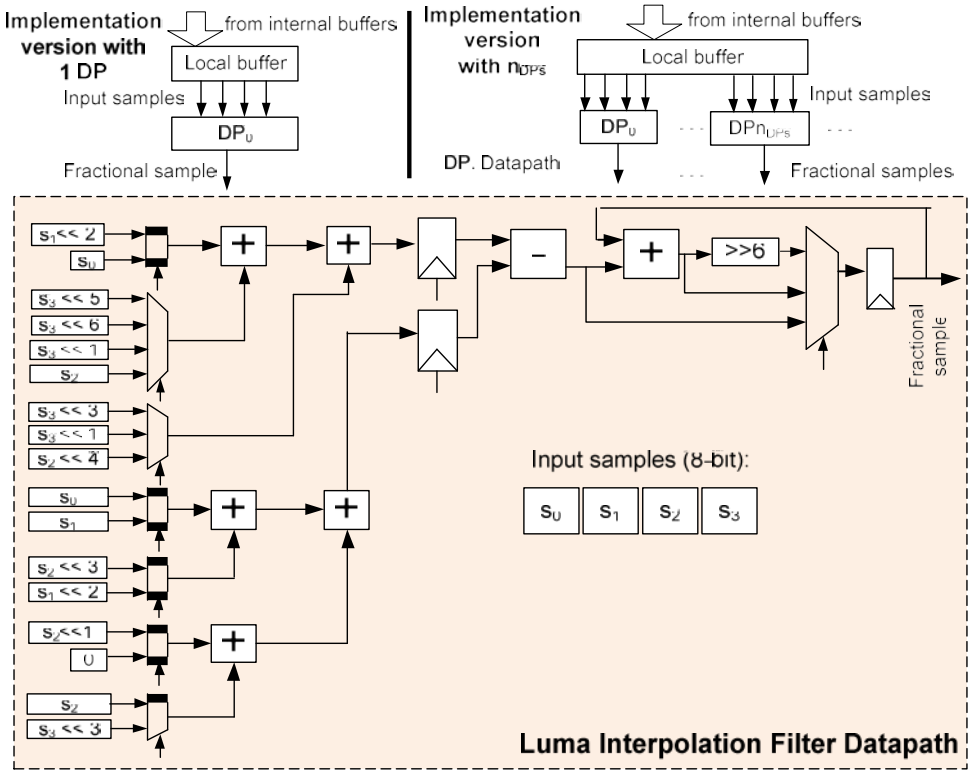
GOP of an Intra-period). For all the other pictures, we apply the first type of prediction, i.e. Equation 5.1. Prediction is performed separately for luma and chroma.

To reduce the prediction error along time, we back-propagate a percentage of the error to the prediction value. The strength of back-propagation is defined by the design-time parameters δ_1 and δ_2, which are selected from offline simulation of a set of video sequences. The set of video sequences are offline simulated with the prediction scheme, and the prediction error is calculated for each picture of each video sequence. The Mean Squared Error (MSE) of each video sequence is computed. Values of δ_1 and δ_2 are adjusted empirically to reduce the MSE of prediction for this set of video sequences.

5.2 Reconfigurable Hardware Engines for Interpolation Filter

We have developed two reconfigurable hardware accelerator engines for luma and chroma interpolation filters. Figure 5.3 illustrates the datapath templates of both engines. We exploit the concept of providing different implementation versions at compile-time for a function to be accelerated (BAUER, 2008). The implementation versions exhibit different requirements of silicon area and performance. They are selected and reconfigured at run-time, as shown in section 5.3. Our datapath templates are scalable and provide the instantiation of 1 to max(n_{DPs}) basic datapaths in parallel. n_{DPs} is the number of datapaths in parallel of a particular implementation version. max(n_{DPs}) is the maximum number of datapaths in parallel that can be instantiated by any implementation version. It must be determined at design-time by the designer considering the worse-case throughput requirements. Each engine also contains local buffers to store input data of datapaths.

Figure 5.3 – Architectural template of the reconfigurable engines. Luma and chroma datapaths are shown in section 4.2. Luma datapath is shown here as an example.



Source: (DINIZ, 2015a).

Luma and chroma datapaths are the same datapaths used in our proposed dedicated hardware architecture for HEVC interpolation filter, shown in section 4.2. Four input pels (s_0-s_3) are given as input. Luma datapath generates one fractional-pel of 7-/8-tap filter at each 2 cycles. Instead, chroma datapath generates one fractional-pel per cycle. This way, an implementation version of the luma reconfigurable hardware engine with n_{DPs} datapaths has an equivalent throughput of $n_{DPs}/2$ fractional-pels per cycle. An implementation version of the chroma engine with n_{DPs} has throughput of n_{DPs} fractional-pels per cycle. Local buffer is designed to store necessary input data to process the implementation version with the highest parallelism.

5.3 Implementation Version Selection

This module selects one out of N different implementation versions (section 5.2) for interpolation filter acceleration. The selection is based on the predicted number of interpolation filter calls, i.e., PNFC (section 5.1). An optimal selection is the one that finds the implementation version with the lowest number of datapaths that satisfies the required performance. The optimal selection consumes less area and saves (leakage) energy. The Estimated Performance (EP) in pixels/picture of a particular implementation version $i \in \{1, ..., N\}$ is given by Equation (5.5). In Equation (5.5), f is the frequency of the reconfigurable engine obtained after synthesis (see Section 5.5). n_{DPs} is the number of datapaths (that process in parallel) of the implementation version i.

$$EP(i) = \begin{cases} \left\lfloor f * n_{DPs}(i) \right\rfloor \left\lfloor 2 * frame\_rate \right\rfloor & \text{if Luma} \\ \left\lfloor f * n_{DPs}(i) \right\rfloor \left\lfloor frame\_rate \right\rfloor & \text{if Chroma} \end{cases} \tag{5.5}$$

Equation (5.6) is used to select an implementation version S_1 with $n_{DPs}(i)$ datapaths that produces an EP higher than the estimated prediction given by PNFC. Apparently, it satisfies both area and performance requirements. However, the prediction may incur an error (see Equation 5.2), and the monitored number of filter calls may be higher than the predicted. Consequently, this selection method may not guarantee performance. The required throughput can be achieved by choosing an implementation version S_2 (Equation 5.7), which has only one additional datapath than S_1 (Equation 5.6).

$$S_1(p,g) = i, \begin{cases} n_{DPs}(i) = \left\lfloor \dfrac{\lceil 2 * PNFC(p,g) * frame\_rate \rceil}{f} \right\rfloor & \text{if Luma} \\ n_{DPs}(i) = \left\lceil \dfrac{PNFC(p,g) * frame\_rate}{f} \right\rceil & \text{if Chroma} \end{cases} \tag{5.6}$$

$$S_2(p,g) = i, \quad n_{DPs}(i) = n_{DPs}(S_1(p,g)) + 1 \tag{5.7}$$

Equation (5.7) always selects an implementation version with slightly higher parallelism at the cost of increased area and leakage energy. To address this issue, Equation (5.8) considers the maximum prediction error max(ε) that may occur at a certain point in time. For each picture p of GOP g, we calculate S_1 (Equation 5.6) and $EP(S_1)$ (Equation 5.5) based on the PNFC(p,g) from the prediction scheme. PNFC(p,g) is then subtracted by $EP(S_1)$. If the result is lower than the maximum error, the selected implementation version with S datapaths is calculated as S_1 (Equation 5.6).

$$S(p,g) = \begin{cases} S_1(p,g) & \text{if } (EP(S_1) - PNFC(p,g) < \max(\varepsilon)) \\ S_2(p,g) & \text{if } (EP(S_1) - PNFC(p,g) >= \max(\varepsilon)) \end{cases} \tag{5.8}$$

For example, in case the PNFC for luma is 40 million, the frame rate is 30 fps, and the operation frequency f is 283 MHz, an implementation version with 10 luma datapaths in parallel would be selected as being the best tradeoff.

To evaluate the performance impact of our architecture due to an error of prediction, we defined the concept of Tolerable Error (TE). TE is the maximum error that can be tolerated by our reconfigurable engine. It is defined by the difference between the EP values of two implementation versions i_1 and i_2 that differ in only one datapath, i.e., $EP(i_1)=n_{DPs}$ and $EP(i_2)= n_{DPs-1}$. Equation (5.9) shows the TE calculation after applying Equation (5.5). We assume that all implementation versions work at the same frequency.

$$TE = \begin{cases} f / 2 * frame\_rate & \text{if Luma} \\ f / frame\_rate & \text{if Chroma} \end{cases} \quad (5.9)$$

This way, max(ε) must be lower than TE, otherwise, even S_2 (Equation 5.7) does not guarantee required performance. By means of our experimental results (Section 5.5) we show this holds true after 2 GOPs, when prediction scheme has sufficient input information (i.e., monitored history of the interpolation filter calls) to provide good prediction result.

5.4 Adaptive Scheduling

Once the accelerator implementation versions are selected, the processing order of fractional-pel is determined by the Adaptive Scheduling scheme. This scheduling scheme is a generalization of the one discussed in section 4.2, since now the number of datapaths in parallel in the accelerating engines are not fixed, but may vary from one picture to another. The pseudo-code of the adaptive scheduling scheme is shown in Figure 5.4. It adapts to the different processing behavior of the interpolation filter depending upon it usage for FME or MC. In FME, it adapts to half-pel and quarter-pel computation.

In MC, it schedules fractional-pels calculation according to the fractional-precision MV received as an input. In general, our scheme calls the schedule function for each case, prioritizing the computation of fractional-pels of the same row or column in parallel, whether filters are applied in horizontal or vertical direction, respectively. In this way, our scheme reuses input data.

The pseudo-code of the schedule function is depicted in Figure 5.5. This function abstracts input data fetching and reconfiguration of interpolation filter types, i.e., fetch and reconfigure functions. It considers the size of PU and the selected implementation version S(p,g) to reconfigure datapaths accordingly. The fetch function is used to read the input data from the internal buffers (as shown in Figure 5.1). Input data may either be integer-pels or fractional-pels. Integer-pels are fetched from external reference picture memory. Fractional-pels a, b and c (see fractional pixel positions in section 2.3.1) are stored into internal buffers since they are used to calculate other fractional-pels. The reconfigure function takes into consideration the $n_{DPs}(S(p,g))$ that defines the number of datapaths in parallel of the selected implementation version S(p,g). The configuration of filter types involves choosing the appropriate input data to the multiplexers of datapaths. When PU_width and PU_height is lower than the number of datapaths in parallel S(p,g) the remaining datapaths can be used to process other samples. Figure 5.6 shows an example of the proposed adaptive scheduling.

Figure 5.4 – Pseudo-code of the adaptive scheduling scheme (for luma engine)

```
Adaptive Scheduling ( )
Inputs:   FME_flag       : a flag to indicate if it is FME
          quarter_flag   : a flag to indicate quarter refinement
          PU_width       : the width of PU (in pixels)
          PU_height      : the height of PU (in pixels)
          MV[x,y]        : the motion vector
Output: order of schedule function calling
begin
   if (FME_flag = 1)    // Interpolation for FME
          if (quarter_flag = '0')   // half-pel refinement for FME
                              for row = -3..PU_height+4
                    schedule frac.-pels b (type 2)
                    for col = -3..PU_width
                              schedule frac-pels h (type 2)
                              schedule frac-pels j (type 2)
          else   // quarter-pel refinement
                              for row = -3..PU_height+4
                                        schedule frac.-pel calc. of types a, and c
                              for col = -3..PU_width+4
                              if best match is in b position
                                                  schedule fractional-pel calc. of types p, e, q, f, r, g
                              if best match is in h position
                                                  schedule fractional-pel calc. of types g, k, r, d, n, e, i, p
                              if best match is in j position
                                                  schedule fractional-pel calc. of types e, i, p, f, q, g, k, r
   else        // Interpolation for MC
                    xFrac = MV[0] && 3;
                    yFrac = MV[1] && 3;
                    if (xFracL != '0') // green and yellow samples (see Fig. 1)
                              for row = -3..PU_height+4
                                        schedule frac.-pel calc. of type [xFracL,yFracL]
                              if (yFracL != '0') // yellow samples (see Fig. 1)
                              for col = -3..PU_width+4
                                                  schedule fractional-pel calc. of type [xFracL,yFracL]
                    else // orange samples (see Fig. 1)
                              for col = -3..PU_width+4
                                                  schedule fractional-pel calc. of type yFracL
end
```

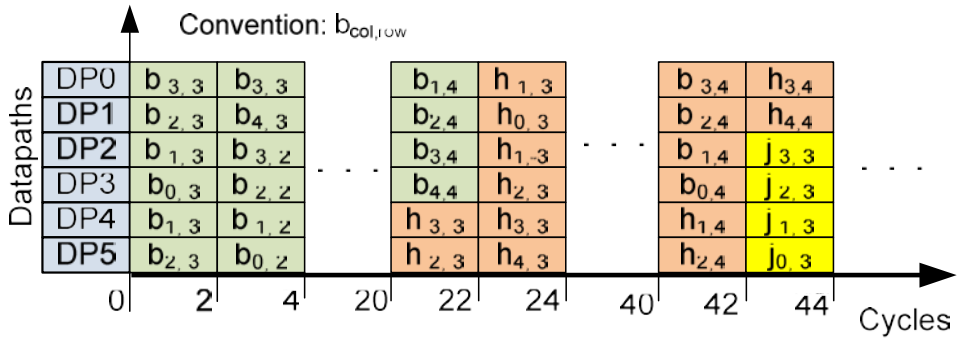
Source: (DINIZ, 2015a).

Figure 5.5 – Pseudo-code of the schedule function

```
schedule ( )
Inputs:   int-pel[]        : a block of integer-pels (internal buffer)
          frac-pel[]       : a block of fractional-pels (internal buffer)
          PU_width         : the width of PU (in pixels)
          PU_height        : the height of PU (in pixels)
          S(p,g)           : the selected implementation version
          types[S(p,g)]    : a vector with the interpolation types
Output: Reconfiguration of DPs.
begin
   if ( types[] are a, b or c ) // horizontal filters
      fetch PU_width+7 of row from int-pel[]
      if ( PU_width > S(p,g) )
         for i = 1…ceil(PU_width / S(p,g) )
           reconfigure n_DPs(S(p,g)) DPs with corresponding types[]
      else
          reconfigure PU_width DPs with corresponding types[]
   else // vertical filters
      fetch PU_height+7 of col from int-pel[] (d,h,n) or frac-pel[]
      if ( PU_height > n_DPs(S(p,g)) )
         for i = 1…ceil(PU_height / S(p,g) )
           reconfigure n_DPs(S(p,g)) DPs with corresponding types[]
      else
       reconfigure PU_height DPs with corresponding types[]
end
```

Source: (DINIZ, 2015a).

Figure 5.6 – Example of scheduling for S(p,g) = 6 and PU_width = 8



Source: (DINIZ, 2015a).

5.5 Results and Evaluation

The reconfigurable hardware engines of our architecture were designed in VHDL and synthesized for the Xilinx XC5VLX110T-2ff1136 FPGA device that supports partial dynamic reconfiguration. Synthesis was performed with the Xilinx ISE synthesis tool. Prediction, Implementation Version Selection and Adaptive Scheduling modules were implemented in software and integrated into the HM (HM, 2013). The detailed results and discussions are shown in the next sections.

5.5.1 Fairness of comparison

Our work provides fully standard compliant filter implementation, thus producing the video quality that matches exactly that of the reference software. Moreover, our architecture provides a standard interface which can easily be used by a fast motion search algorithm, thus not limiting its applicability. Approaches that employ fast mode decision and search algorithm may exhibit quality loss compared with our work. We employed the same ME and mode decision schemes as those being used in the reference software (HM, 2013). Therefore, our comparison results only demonstrate the effects of fast architecture on performance/throughput, power, and area results.

We determined the throughput values of our architecture using full, detailed logic system simulations (containing the function accurate model of the hardware architecture) for the operating frequency obtained after the synthesis. For instance, for the worst case scenario, we executed a set of 2560x1600 videos. The input to the implementation selection mechanism is the predicted number of filter calls (PNFC) and the output of this analysis is the number of datapaths to be reconfigured for each picture. The maximum number of datapaths selected considered all pictures of all videos for the 2560x1600 resolution was 17.

5.5.2 Synthesis Results

Table 5.1 shows the synthesis results of the reconfigurable hardware accelerator engines given a worst-case scenario to process videos of resolution = 2560x1600.

It employs 17 DPs, since our scheme selects implementation versions for ultra-high resolution videos with n_{DPs} in which $\max(n_{DPs})=17$. The luma engine operates at 283 MHz frequency, and the chroma engine operates at 184 MHz. The luma and chroma local buffers store 71 and 35 input samples, respectively. By means of our adaptive

scheduling scheme, it is sufficient to store input data to interpolate one row or column at a time for a PU of size up to 64x64.

Table 5.1 – Synthesis results of the proposed hardware architecture for the worst-case throughput constraint (i.e., 2560x1600 @ 30 fps). Consider max(n_{DPs})=17.

Module	Slice LUTs	Slice registers	Occupied Slices	BRAMs	Dynamic Power (mW)	Reconfiguration Time[2] (µs)
Luma local buffer	-	-	-	1	2	-
Luma datapaths[1]	2,602	1,513	1,131	-	47	636.51
Chroma local buffer	-	-	-	1	2	-
Chroma datapaths[1]	2,415	1,037	1,050		38	590.93
Total	5,017	2,550	2,181	2	89	1,227.44

1. max(n_{DPs})=17.
2. Estimated based on Occupied Slices and considering a reconfiguration bandwidth of 3.2 Gbps (ICAP with 32-bit operating at 100 MHz (XILINX, 2012)).

Source: (DINIZ, 2015a).

The pure software implementation of interpolation filter for an 8x8 block on the Leon-II requires 6336 cycles. However, our hardware architecture requires only 160 cycles that correspond to a speedup of 39x.

Our approach supports reconfiguration of different architectural templates with different number of datapaths to support diverse coding configurations, i.e., resolutions, and frame rate (see Table 5.2). Each of these architectural templates with a given number of datapaths supports the maximum requirements for the corresponding coding configuration. Since our architecture is scalable, the supported throughput requirements of the architecture can be decided at design-time. Table 5.1 shows a design case to support 2560x1600@30 fps. Table 5.2 shows another design case with max(n_{DPs})=6 to support 832x480@60 fps. The design for 832x480@60 fps requires 65% less Look-up tables (LUTs) than the design for 2560x1600@30fps. In case the system requirements denote smaller resolutions, a small-sized FPGA may be deployed that saves area. However, for a given coding configuration, sufficient area should be provided. In addition to that, our architecture saves area by applying design optimizations to reduce the number of operations to support the three types of filters in the reconfigurable filter datapaths of the luminance engine.

Table 5.2 shows the synthesis results of six different implementation versions (with n_{DPs} varying from 1…6) for the same FPGA device. These versions provide the required throughput to encode video sequences of resolution 832x480 pixels. At run-time, our technique saves dynamic power for changing workload scenarios, while the total area is fixed at design-time for the worst case of a given coding configuration. We estimated the dynamic power consumption of our architecture for FPGA using the Xilinx Power Estimator (XPE) tool (XILINX, 2013). In case our architecture is mapped to a low power FPGA that supports power-gating (TUAN, 2006) we can also achieve leakage power reduction.

Table 5.2 – Synthesis results of six implementation versions for luma and chroma hardware acceleration engines

	n_{DPs}	Number of Interpolated Pixels	Slice LUTs	Slice registers	Occupied Slices	Reconfiguration Time[1] (µs)	Dynamic Power (mW)
Luma engine	1	4,716,667	154	89	60	33.77	4
	2	9,433,333	307	178	131	73.73	8
	3	14,150,000	460	267	167	93.99	12
	4	18,866,667	613	356	266	149.70	16
	5	23,583,333	766	445	333	187.41	20
	6	28,300,000	919	534	370	208.23	23
Chroma engine	1	9,433,333	143	61	59	33.20	2
	2	18,866,667	285	122	118	66.41	4
	3	28,300,000	427	183	177	99.61	7
	4	37,733,333	569	244	237	133.38	9
	5	47,166,667	711	305	296	166.59	11
	6	56,600,000	853	366	356	200.35	13

1. Estimated based on Occupied Slices and considering a reconfiguration bandwidth of 3.2 Gbps (ICAP with 32-bit operating at 100 MHz (XILINX, 2012)).

Source: (DINIZ, 2015a).

5.5.3 Discussion on Reconfiguration Latency

To provide such flexibility to adapt to different throughputs, our architecture incurs reconfiguration latency, as shown in Tables 5.1 and 5.2. However, this is only a startup time reconfiguration when switching between different coding configurations on a picture-by-picture basis. In the following, we show that the reconfiguration time is negligible compared to the full frame encoding time. It can even be hidden using partial reconfiguration (XILINX, 2010) (ALTERA, 2010) and configuration prefetching (LI, 2002).

The reconfiguration time for the real-time encoding scenarios is estimated as follows. The total number of configuration bits of our target FPGA device (XC5VLX110T) is 31,118,848 bits (XILINX, 2012). Considering that the reconfiguration is performed through the Xilinx Internal Configuration Access Port (ICAP) operating at 100 MHz with 32-bit interface (XILINX, 2012), the reconfiguration bandwidth is 3.2 Gbps. Therefore, the time to reconfigure the entire device (17,280 Slices) is 9.725 ms. A particular implementation version corresponds to a partial bitstream to be loaded using PRR. We estimated the reconfiguration time of the partial bitstream by Equation (5.10) where Number_of_occupied_Slices is the number of slices occupied by a particular implementation version with n_{DPS} (see Tables 5.1 and 5.2).

$$\text{Reconfiguration\_time} = \frac{\text{Number\_of\_occupied\_Slices} * 9.725\text{ms}}{17280} \tag{5.10}$$

Note, in our work, the reconfiguration of a particular implementation version is performed only once per encoded picture. In a real-time encoding scenario, each picture needs to be processed within 16.67 ms for a 60 fps scenario. Considering the worst case condition, i.e., to reconfigure the implementation version with 17 DPs for both luma and chroma engines (the worst case for 2560x1600 resolution) the reconfiguration time consumes 1.23 ms (as shown in Table 5.1) that corresponds to only 7.3% of the total

time to encode 2560x1600@ 60 fps video. Since our architecture is adaptive, for some pictures it may require a lower reconfiguration time, e.g.,

- 66 µs, if nDPS=1 for both luma and chroma engines (33 µs for luma and 33 µs for chroma). It represents only 0.39% of the picture encoding time at 60 fps.
- 408 µs, if nDPS=6 for both luma and chroma engines (208 µs for luma and 200 µs for chroma). It represents only 2.44% of the picture encoding time at 60 fps.

5.5.4 Discussion on Reconfiguration Energy

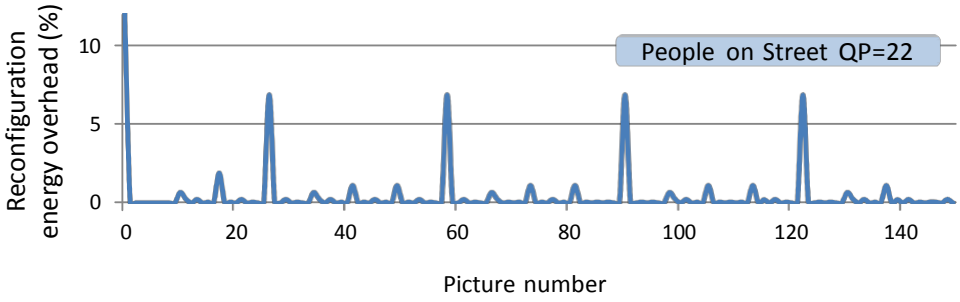
Since the available FPGA power estimation tools (XILINX, 2013) do not provide reconfiguration power, we use the data provided by the work in (BECKER, 2010) that measured the reconfiguration power of a Xilinx Virtex-5 FPGA device using the Microblaze processor and ICAP, the same as we used in this work. The work in (BECKER, 2010) measured the reconfiguration power of a Virtex-5 FPGA device by measuring the reconfiguration current of a modified Xilinx ML505 FPGA board that includes a precision current sense resistor between the voltage regulator and the FPGA core supply, with the help of a digital storage oscilloscope. The measured reconfiguration power to reconfigure 12,867 LUTs on a Virtex-5 device, in their experiment was 0.44 W (BECKER, 2010). This experimental data from these authors was used herein for estimations of reconfiguration power cost. Therefore, the estimated power to reconfigure our worst case design (for 2560x1600 resolution, consuming 5,017 LUTs) is 0.17 W. The worst case reconfiguration energy is given as 0.21 mJ (i.e., 0.17 W * 1.23 ms; consider the reconfiguration latencies shown in Table 5.1 and Table 5.2). For this case, the processing energy is given as 1.48 mJ (89 mW * 16.67 ms) to interpolate one frame at a frame rate of 60 fps. This shows that, in the worst case, the reconfiguration energy contributes towards 12% of the total energy. However, this is done for only once, if the picture requires 17 datapaths in parallel for both luma and chroma. If the subsequent pictures also require 17 datapaths, then no (additional) reconfiguration is performed.

We now analyze two normal cases, as we showed in the reconfigurable time discussion. In case 1, one datapath of luma and one datapath of chroma (297 LUTs) need to be reconfigured. It requires a reconfiguration time of 66 µs for each picture. The power to reconfigure 297 LUTs is 10.15 mW. The reconfiguration energy in this case is 0.67 µJ (i.e., 10.15 mW * 66 µs). As the processing energy is 100 µJ (i.e., 6 mW * 16.67 ms), the reconfiguration energy represents only 0.66% of the total energy. The second case is when 6 datapaths of luma and 6 datapaths of chroma (1,772 LUTs) need to be reconfigured. It requires a reconfiguration time of 408 µs for each picture. The power to reconfigure 1,772 LUTs is 60.59 mW (i.e., 60.59 mW * 408 µs). The reconfiguration energy in this case is 24.72 µJ. As the processing energy is 600.12 µJ (i.e., 36 mW * 16.67 ms), the reconfiguration energy represents 3.9% of the total energy.

Considering the average case variations, the net reconfiguration overhead is always below 1.32% only. Figure 5.7 shows the reconfiguration energy overhead per picture (in % of the total energy per picture) for People on Street video sequence (2560x1600 pixels). Further optimizations on reducing the reconfiguration overhead may be performed which are orthogonal to the novel contributions of our work. For instance, lowering the reconfiguration power using the methods of (CLAUS, 2008) will further

reduce our reconfiguration energy overhead to 14.52 µJ, i.e., only 2.36 % of the total energy.

Figure 5.7 – Reconfiguration energy overhead (%)



Source: (DINIZ, 2015a).

5.5.5 Comparison with State of the Art

Table 5.3 shows the comparison of our architecture with state of the art. The most relevant comparison partners are the works in (PASTUSZAK, 2013) (ZATT, 2013) (AFONSO, 2013) as they provide results for FPGA. Since our architecture is scalable, we show results of two versions with different throughputs: (1) for ultra-high resolution 2560x1600 @ 30 fps, which is the higher resolution video provided in (BOSSEN, 2013); and (2) for medium 832x480 @ 60 fps.

Table 5.3 – Comparisons with state of the art hardware architectures for fractional-pixel interpolation filter

	(PASTUSZAK, 2013)[1]	(ZATT, 2013)[1]	(AFONSO, 2013)	This work	
Standard	H.264	H.264	HEVC	HEVC	HEVC
Chroma Interpolation	Yes	Yes	No	Yes	Yes
FPGA device & technology	Aria II GX 40 nm	Virtex-2P 90 nm	Stratix-III 65 nm	Virtex-5 65 nm	Virtex-5 65 nm
Throughput (pixels/cycle)	8	6	27	25.5	9
LUTs	6,732	6,742	4,077 + 1633[4]	5,017	1,772
Registers	-	5,904	3,861	2,550	900
BRAMs	-	0	1[2,3]	2	2
Multipliers	-	8	-	-	-
Resource efficiency (throughput/LUT)*100	0.11	0.08	0.47	0.50	0.50
Dynamic Power for worst case (mW)	60[5]	88[6]	379[5]	89[6]	36[6]
Dynamic power (mW) scaled to 65 nm[7]	85	88	379	89	36
Throughput	1920x1080 @ 30 fps	1920x1080 @ 60 fps	3840x2160 @ 60 fps[2]	2560x1600 @ 30 fps	832x480 @ 60 fps
Throughput/power	0.09	0.06	0.07	0.28	0.25

[1]Results of the Interpolation Filter only. [2]Considers only luma interpolation. [3]Includes 16,547 bits of buffers, which compares to one 18 Kb BRAM. [4]Corresponding to the MUX in (AFONSO, 2013). [5]Estimated with Altera early power estimator tool (ALTERA, 2013). [6]Estimated with Xilinx power estimator tool (XILINX, 2013). [7]Scaled using the power scaling factors derived in (SHAFIQUE, 2014) using data provided in (KLEIN, 2009).

Source: (DINIZ, 2015a).

Since different architectures in related works use different settings which are either incomplete, e.g., no chroma interpolation in (AFONSO, 2013), or for a different

technology, e.g. work in (ZATT, 2013) targets 90nm, a direct power/performance comparison to our architecture is not straightforward. Therefore, we have performed power estimation and power normalization for our work and related works using the same methodology. Power estimation uses Xilinx power estimator tool (XILINX, 2013) and Altera early power estimator tool (POWERPLAY, 2014). Power normalization is based on dynamic power scaling factors between older and new technology FPGAs obtained by the work in (SHAFIQUE, 2014) based on information provided in Xilinx datasheet (KLEIN, 2009). Power scaling factors for 90 nm, 65 nm and 40 nm technologies are 0.4, 0.41, and 0.287, respectively, compared to 150 nm technology. Based on these power scaling factors derived in (SHAFIQUE, 2014), we have normalized to 65 nm the dynamic power results from related work whose technologies differ from our work, i.e. 40 nm (PASTUSZAK, 2013) and 90 nm (ZATT, 2013). Additionally, to have a more fair comparison independent of different resolutions and area values, we compare different architectures for two new metrics: (1) Throughput in terms of pixels processed per cycle (i.e., pixels/cycle); and (2) Resource Efficiency in terms of throughput per area. Table 5.3 shows that our architecture is better compared to state of the art in terms of area, throughput, resource efficiency, and power consumption for different cases. Detailed discussion is shown in the following.

The buffers requirements in (AFONSO, 2013) are 16,547 bits that correspond to one 18 Kb Block RAM in our design. Our design requires 6,112 bits of buffers, which could be mapped to only one BRAM. However, it uses two BRAMs to provide parallel access to luma and chroma input samples to achieve high throughput. Work in (AFONSO, 2013) does not support chroma interpolation (less data need to be stored). If we disable BRAMs in our design, it uses additional 764 LUTs to store samples into registers. The buffers in (AFONSO, 2013) correspond to 2,069 LUTs. Moreover, the design of (AFONSO, 2013) requires 256 4:1 MUX and 1377 2:1 MUX that are mapped to additional 1633 LUTs. Therefore, in total, the design in (AFONSO, 2013) requires 5,710 LUTs. In terms of functionality, our work is still better compared to the work of (AFONSO, 2013) because the latter only supports Luma interpolation consuming 5,710 LUTs while our proposed architecture requires 5,017 LUTs for both Luma and Chroma interpolation, which demonstrates high area efficiency of our work compared to the related work. We provide area comparison in terms of LUT count because it is a widely adopted metric for comparison of different FPGA based designs. Since FPGA devices' Altera LUTs in (PASTUSZAK, 2013) and (AFONSO, 2013) implement all 6-input functions, comparing LUT count with our work (Xilinx 6-input LUT) is a fair comparison. Since the work in (ZATT, 2013) uses a FPGA with 4-input LUT, we have normalized the LUT counts accordingly for a fair comparison.

Regarding throughput comparison, the work in (AFONSO, 2013) have a throughput of 27 luma pixels/cycle at 379mW, while our 2560x1600-pixel version have 25.5 pixels/cycle (i.e., 17 luma pixels are produced at each 2 cycles, which is equivalent to 8.5 pixels/cycle; and 17 chroma pixels/cycle are produced) at only 89 mW.

The comparison with related work can also by conducted in terms of resource efficiency. This metric, as shown in Table 5.3, is computed as (pixels/cycle)*100/LUT. Our architecture provides a resource efficiency of 0.50 while the work in (AFONSO, 2013) leads to a resource efficiency of 0.47. It shows that our architecture brings 6% improvement in the resource efficiency while also providing the complete interpolation functionality (Luma and Chroma) compared to (AFONSO, 2013) that only supports Luma interpolation. Note that the work in (AFONSO, 2013) synthesizes the architecture

to an 8-input LUT Altera Stratix-III FPGA, while our work uses a 6-input LUT Xilinx Virtex-5 FPGA, which further demonstrates improved design efficiency of our architecture.

The work in (AFONSO, 2013) achieves high throughput using (1) deep pipeline (4-stage pipeline) that consumes 3,861 registers, i.e., a 39% increase compared to our work; and (2) high operating frequency (403 MHz), i.e., a 70% increase compared to our work. As a result it incurs a much higher dynamic power (379 mW) compared to our work (89 mW), i.e., 4.25x higher compared to the power consumed by our architecture in the worst case scenario.

Table 5.4 provides comparisons for the worst-case dynamic power for the same throughput (i.e. same video resolution and frame rate) with respective state-of-the-art. We achieve 32% power reduction compared to (PASTUSZAK, 2013) and around 1% power increase compared to (ZATT, 2013) (both implement the 2x less complex H.264 interpolation filter). Our design (processing both Luma and Chroma) achieves a worst-case dynamic power saving of 7% compared to the design of (AFONSO, 2013) (processing only Luma) for the same frame resolution and frame rates. Practically, our savings would be more than 7% compared to (AFONSO, 2013) when considering both Luma and Chroma.

When processing 2560x1600 video, the work in (AFONSO, 2013) would use 379 mW of power due to its non-adaptivity, but using 4x less time, i.e. 0.25 second to process 30 frames. Hence, its energy consumption is equal to 94.75 mJ for processing 2560x1600 @ 30fps. However, in the worst-case (i.e., when always using 17 datapaths), our architecture consumes 89 mJ to process 2560x1600 @ 30fps.

Table 5.4 – Worst-case dynamic power (mW)[1] comparison with state of the art for the same throughput

	Throughput (resolution and frame rate)		
	1920x1080@30fps	1920x1080@60fps	3840x2160@60fps
(PASTUSZAK, 2013)	85 mW	-	-
(ZATT, 2013)	-	88 mW	-
(AFONSO, 2013)	-	-	379 mW
This work	58 mW	89 mW	356 mW

[1]Scaled using the power scaling factors derived in (SHAFIQUE, 2014) using data provided in (KLEIN, 2009).

Source: (DINIZ, 2015a).

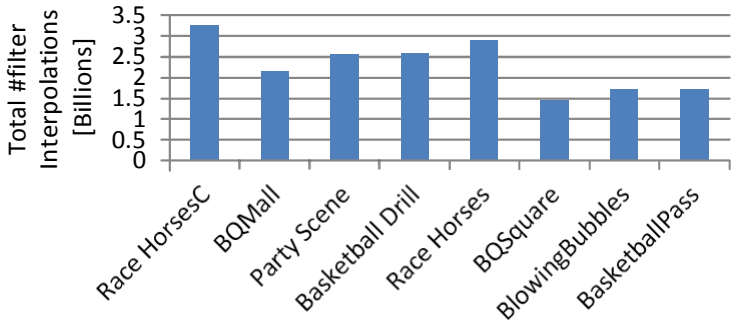
Our architecture still keeps all the benefits of adaptivity, i.e., our work is capable to adapt to different coding scenarios with resolutions smaller than 2K in a power efficient way as demonstrated in the last two columns of Table 5.3. In contrast, the related works always consumes a high power, i.e. of (AFONSO, 2013) consumes 379 mW. Our architecture provides more energy savings under varying workload scenarios (see section 5.5.9). Therefore, the throughput/power efficiency of our design is higher compared to that of state-of-the-art techniques. Our flexibility, power-efficiency, and completeness w.r.t. providing both Luma and Chroma enables our architecture to be preferable in practical scenarios.

5.5.6 Performance Results for Different Video Sequences

In this section we analyze the performance of our reconfigurable architecture for different videos and QPs. Figure 5.8 shows the total number of filter interpolations that our architecture performs to encode 300 pictures of each video sequence. Each bar

shows the average over results extracted with four QPs (22, 27, 32, and 37). Unlike state-of-the-art architectures, Figure 5.8 shows that our architecture adapts its performance depending upon the processing requirements of different video sequences. As motivated in chapter 3, the number of video calls varies depending on video content and our architecture adapts its performance to this variation.

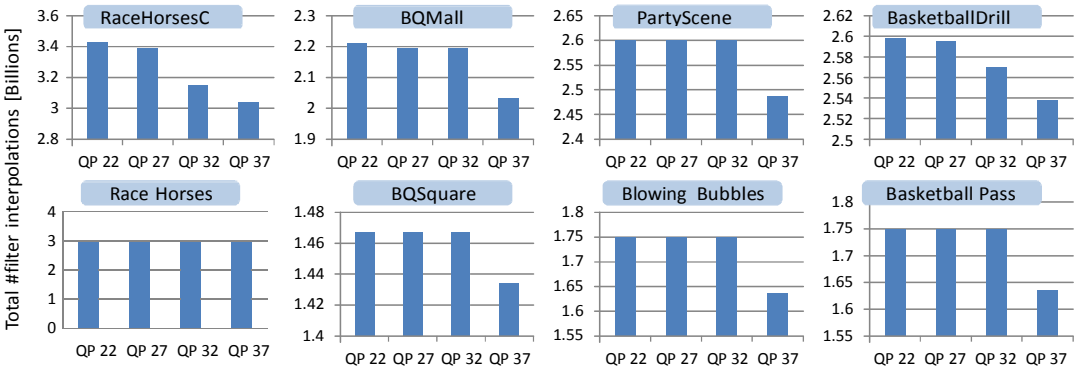
Figure 5.8 – Number of total filter interpolations of our architecture for the set of test video sequences (averaged over QPs)



Source: (DINIZ, 2015a).

A more detailed performance analysis is depicted in Figure 5.9. It shows the total number of interpolations calculated by our architecture for each video sequence, detailing the results for each QP. Figure 5.9 shows that our architecture adapts also to the variations on QP even for the same video sequence. WQVGA video sequences (Race Horses, BQSquare, Blowing Bubbles and Basketball Pass) do not exhibit significant variations over QP changes because the implementation version with 1 DP in parallel is often selected by our Implementation Version Selection scheme. This implementation version is enough to provide the required performance. However, for some WVGA video sequences the adaptation is more evident when comparing different QPs in the same video sequence. Not all sequences exhibit significant variations in the interpolation filter calls for different Quantization Parameter (QP) values.

Figure 5.9 – Number of total filter interpolations of our architecture for each video sequence (for each QP)



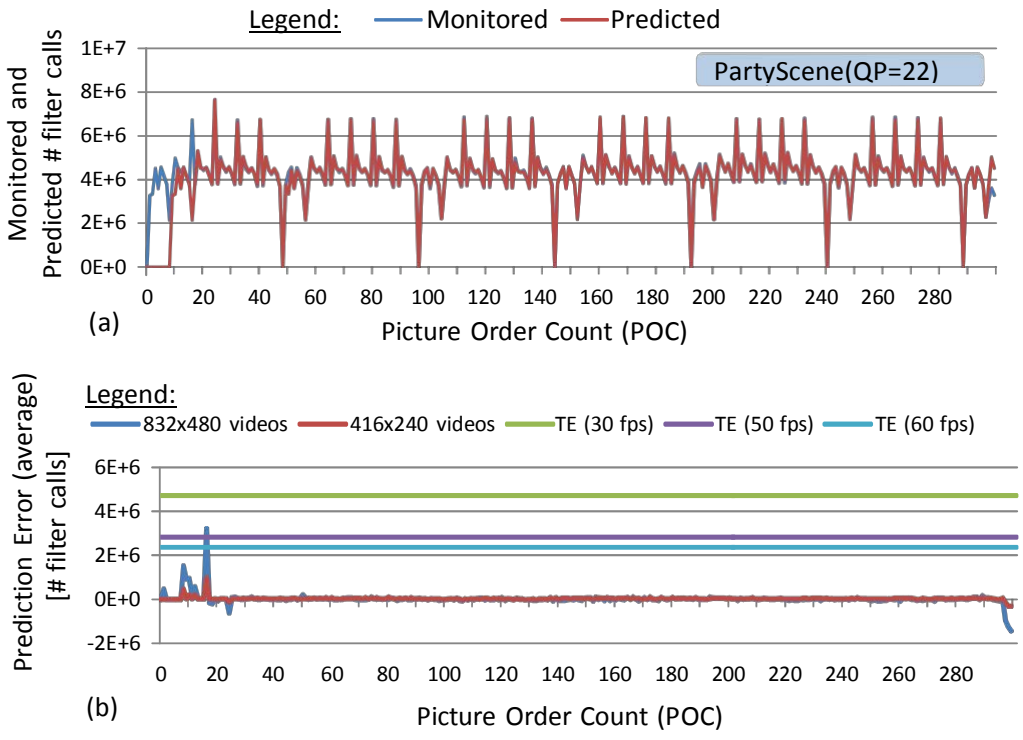
Source: (DINIZ, 2015a).

As we motivated in chapter 3, the sensitivity of variations and QP values is due to many reasons like motion/texture content and the IME algorithm. For low-to-medium motion video sequences such as Basketball Drill (the camera remains still while the basketball players are moving), IME already finds a good match for many blocks, so the total number of interpolation calls is lower. For another case, Race Horses is a complex video scene where a group of horse riders move around a grassy background and the camera also moves, following their motion. Therefore, in this case, the total filtering interpolation is the highest among all video sequences, as the FME is needed more frequently. An efficient interpolation filter architecture needs to take care of above-discussed factors. Therefore, we evaluate our proposed architecture for all these different types of test videos and demonstrate our benefits for all cases.

5.5.7 Evaluation of Prediction Results

Our Prediction Scheme is implemented into HM (HM, 2013) and evaluated for various video sequences. Figure 5.10a shows monitored and predicted values along with the prediction error using δ_1=0.2 and δ_2=0.05. These weight values reduced MSE for the tested video-set that includes 832x480-pixel and 416x240-pixel resolutions (see Figure 5.10b). The prediction error in Figure 5.10b is an average of the videos and the four different QP values. Figure 5.10a shows that our scheme starts the prediction after monitoring one GOP. It improves the estimation after concluding one Intra-Period. As from the second Intra-Period it could apply both prediction types that we proposed in Section 5.1. By applying both prediction types, the average prediction error is well below the Tolerable Error (TE) for three different frame-rates.

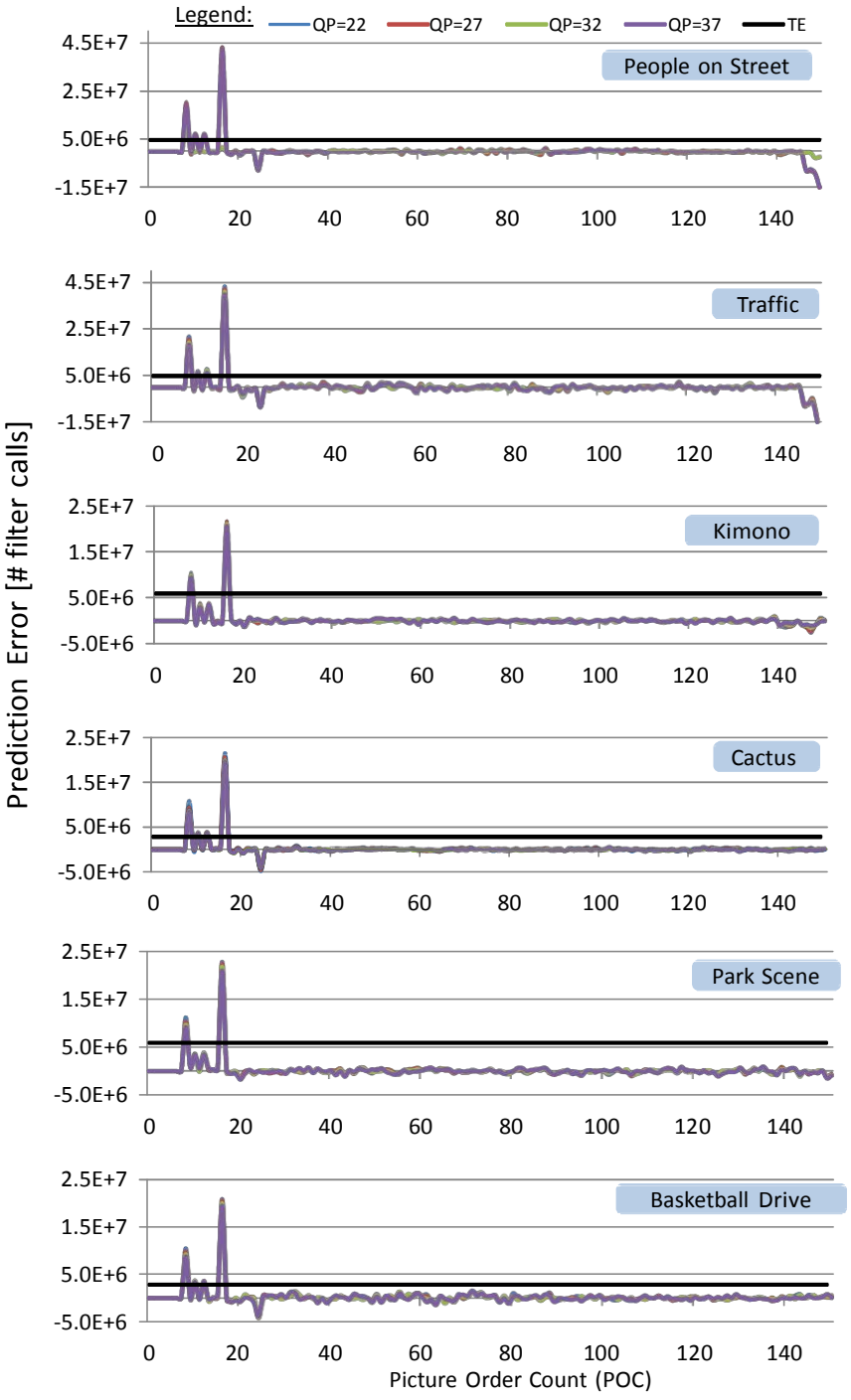
Figure 5.10 – (a) Monitored and predicted number of filter calls; (b) Prediction error for 832x480 and 416x240 video sequences



Source: (DINIZ, 2015a).

In order to illustrate the generality of our concept and to validate that our offline selected parameters are equally applicable to other video sequences, we present in Figure 5.11 the results of prediction error using another 6 video sequences of 2560x1600-pixel resolution “Traffic” and “People on Street” and 1920x1080-pixel resolution “Kimono”, “Park Scene”, “Cactus” and “Basketball Drive”. These sequences are encoded according to the common test conditions (as recommended by the standardization committee) with four different QPs={22,27,32,37}.

Figure 5.11 – Prediction error for 2560x1600 and 1920x1080 videos, considering four QP values



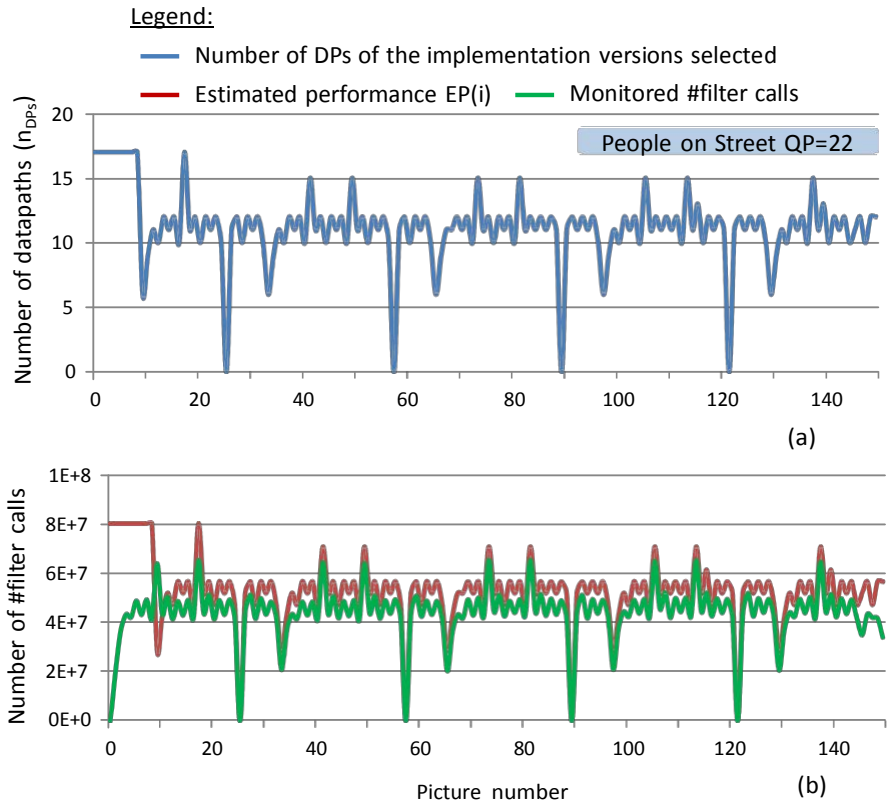
Source: (DINIZ, 2015a).

Note that these video sequences are not used to determine the parameters δ_1 and δ_2, rather for only evaluation. These new results demonstrate that the prediction error is also well below the TE.

5.5.8 Evaluation of Run-time Implementation Version Selection

Figure 5.12 shows the detailed results of our run-time implementation version selection scheme. It shows how our architecture adapts to different throughputs on a picture-by-picture basis based on the adaptive prediction. Different throughputs are achieved by the varying number of datapats of the selected implementation versions, as shown in Figure 5.12a. In the first GOP, as the prediction may incur in errors, we guarantee throughput by selecting the highest throughput implementation version (with 17 datapaths in this case). For the following GOPs, implementation versions with 6 up to 17 datapaths are selected depending upon the frame-rate and frequency (fixed at design-time), and the estimated prediction value at run-time. Figure 5.12b compares the estimated performance (EP) of the selected implementation versions with the monitored number of interpolation filter calls. This figure shows that our architecture guarantees the performance for all pictures after the first Intra-Period (when Prediction accuracy is enhanced) and adapts at run-time to the number of filter calls for each picture.

Figure 5.12 – Implementation version selection results: (a) Number of DPs of implementation versions selected; (b) Comparison of Estimated Performance (EP) of implementation version selected and the monitored number of filter calls for each picture (FME case, luma interpolation filter)



Source: (DINIZ, 2015a).

5.5.9 Comparison with a Non-Reconfigurable Implementation

In this section, we compare two solutions (all using on FPGA-based designs):

- Case 1 – All 17 datapaths are available and always powered-on and no reconfiguration is performed: In this case the system will consume more dynamic power but saving the reconfiguration energy.
- Case 2 – All 17 datapaths are available and unused datapaths are power-gated. In this case, power-gating will incur the loss of reconfiguration data (SHAFIQUE, 2009) (SHAFIQUE, 2010). Therefore, before using the additional datapaths again, additional reconfigurations need to be performed. In this case, the energy of unused datapaths is saved at the cost of additional reconfiguration energy. We in fact do consider such a case for our work. Since power-gating is not currently available in the commercial FPGAs, we aligned our discussion more towards the reconfiguration and dynamic power/energy.

Let us now compare the energy consumption of two cases for a scenario requiring 6 datapaths of luma and 6 datapaths of chroma (1,772 LUTs); see summary of results in Table 5.5. In Case 1, since the system does not support reconfiguration of additional datapath, all 17 datapaths are made always available and therefore are always kept powered-on. In this case, the total energy consumption is given as 1.48 mJ (i.e., 89 mW * 16.67 ms). In Case 2, we consider both the energy to reconfigure the 6 datapaths of luma and 6 datapaths of chroma and the processing energy of those datapaths for a picture. The reconfiguration energy is given as 24.72 µJ (i.e., 60.59 mW * 408 µs) and the processing (dynamic) energy is given as 600.12 µJ (i.e., 36 mW * 16.67 ms). It results in total energy of 624.84 µJ that represents an energy saving of 58% for our case compared to the non-reconfigurable solution.

Table 5.5 – Comparison with non-reconfigurable design

	Non-reconfigurable	Our
Reconfiguration energy	-	24.72 µJ
Processing energy	1.48 mJ	600.12 µJ
Total energy	1.48 mJ	624.84 µJ

Source: (DINIZ, 2015a).

6 RUN-TIME ACCELERATOR BINDING INTO RECONFIGURABLE ARCHITECTURES

Section 2.4 discussed some benefits of reconfigurable architectures compared to dedicated architectures. In addition to providing low NRE cost, reconfigurable architectures (especially FPGAs) can enable run-time power/energy saving as discussed in Chapter 5. FPGAs introduce flexibility at bit level through fine-grained reconfiguration, at the cost of higher power dissipation compared to dedicated circuits. On the other hand, coarse-grained reconfiguration provides lower flexibility than FPGA but also lower power dissipation.

Section 2.4 contains a review of a recent trend of mixed-grained reconfigurable processors that integrate several coarse-grained (CG) and fine-grained (FG) reconfigurable elements. Those mixed-grained reconfigurable processors combine the benefits CG and FG reconfigurable elements for an efficient accelerator-rich architecture. In this work, we envision that, with the rise of on-chip reconfigurable fabrics coupled with many core processors, these architectures will be organized in processing tiles, on a single chip, to ensure scalability to many cores. Such architectures require a run-time system to adapt to the application requirements and to manage accordingly the reconfigurations. One important step of the run-time system is the accelerator binding.

This chapter presents and discusses our novel run-time accelerator binding scheme for tile-based mixed-grained reconfigurable processors. It is generic to account for the diverse properties of CG- and FG- reconfigurable elements, their organization inside tiles, and the total communication delay. Given an architectural configuration, our scheme determines a communication-minimizing binding for datapaths of custom instructions at run-time, employing datapath reusing and inter-tile communication cost estimation.

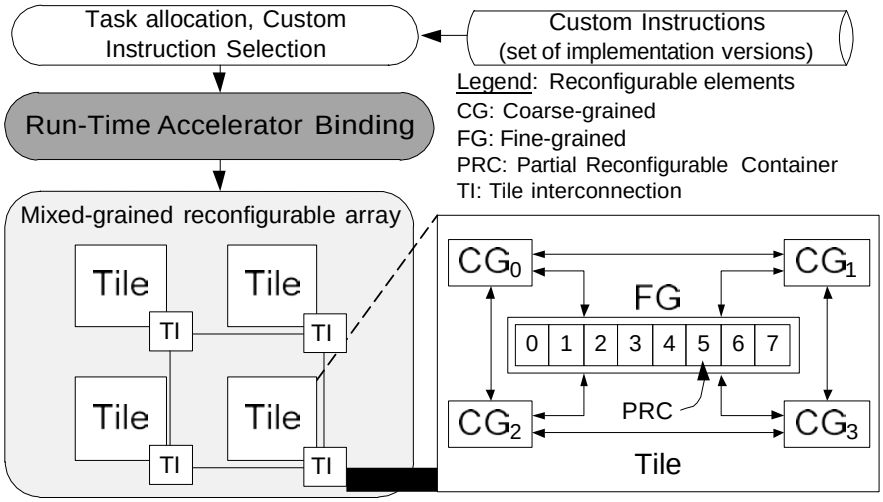
6.1 Overview of Tile-based Reconfigurable Architecture

Figure 6.1 shows the system diagram of our tile-based mixed-grained reconfigurable architecture with our novel run-time accelerator binding module. The Custom Instruction Selection module delivers a set of accelerators that compose custom instructions to our binding module at run time. The internal composition of custom instructions is detailed in (SHAFIQUE, 2011). They are similar to the proposed accelerators for HEVC in this work. This set of accelerators is chosen for each custom instruction from a set of implementation versions (available at design time) depending upon the available reconfigurable area (AHMED, 2011) (AHMED, 2011a). After selection, these datapaths need to be bound/placed and reconfigured onto physical FG- and CG-reconfigurable elements to realize the functionality of the custom instructions.

The mixed-grained reconfigurable processor is composed of multiple tiles. Each tile consists of multiple CG- and FG- reconfigurable elements. The number of reconfigurable elements inside each tile and in the whole architecture is a design time decision. In this paper, we adopt the architecture of CG- and FG- reconfigurable elements of KAHRISMA architecture (KÖNIG, 2010), which illustrate the benefits of their design over existing mixed-grained reconfigurable processors. The FG fabric is partitioned into run-time Partial Reconfigurable Containers (PRC). Each PRC reconfigures one datapath with FG granularity (AHMED, 2011a). The CG-fabric is

composed of configurable ALUs with dedicated register file and context memory (KÖNIG, 2010).

Figure 6.1 – Abstract System Overview: Our proposed run-time accelerator binding module integrated within the tile-based mixed-grained reconfigurable architecture



Source: (DINIZ, 2014).

The reconfigurable elements inside a tile exchange data with dedicated point-to-point interconnections. Different tiles exchange data through a fixed word-width Tile Interconnection (TI) on-chip network. For this reason, the communication between reconfigurable elements of different tiles may spend much more cycles than communicating within the same tile. To achieve high performance, a binding scheme has to prioritize mapping datapaths of the same custom instruction inside the same tile, as motivated in section 6.2.

6.2 Motivational Analysis

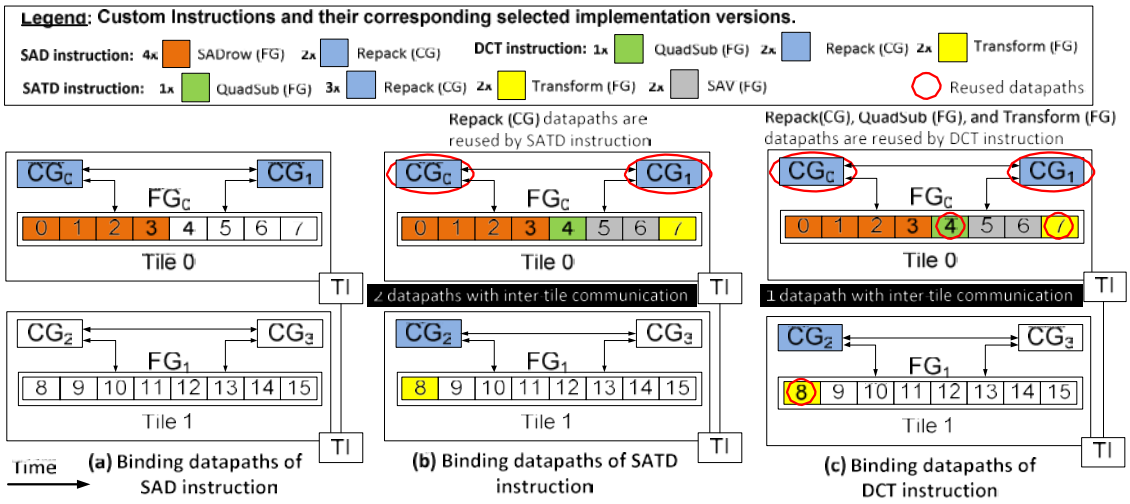
Before proceeding to the details of our novel binding scheme, we present a binding scenario and its implications by means of a H.264/AVC video encoder application.

Figure 6.2 shows an example of binding three custom instructions used to accelerate compute-intensive kernels of the H.264 encoder: (1) Sum of Absolute Differences (SAD), (2) Sum of Absolute Transformed Differences (SATD), and (3) Discrete Cosine Transform (DCT). Each custom instruction is composed of a set of datapaths, e.g., the SADrow datapath perfoms SAD to a row of a 16x16-pixel block in the input video and the Repack datapath performs different byte packing and merging operations. Each datapath can be accelerated either by a CG- or a FG- reconfigurable element. For each custom instruction, a set of implementation versions, with different number of CG and FG datapaths, is available at design-time. An appropriate version is selected at run-time for each custom instruction by the Custom Instruction Selection scheme (AHMED, 2011a). In the example of Figure 6.2, SAD custom instruction implementation version selected has 4 datapaths of SADrow (in FG) and 2 datapaths of Repack (in CG).

Figure 6.2 shows the binding of datapaths of three custom instructions using a first-fit with datapath reusing scheme (BAUER, 2008). In the example, each tile includes 4 CG-elements and 2 FG-elements each containing 8 PRCs. The elements of the architecture are labeled with indexes. To bind a datapath, the first-fit scheme scans the

indexes of the elements from low to high and binds to the first element that is free. This scanning is performed separately for CG and FG, because the datapath must match the granularity of the physical element. Hence, this scheme is not aware of the tile organization shown in Figure 6.2. We assume that the custom instruction selection scheme is aware of the total number of CG and FG elements (AHMED, 2011a). In this way, when a custom instruction is available for binding there are enough elements available in the architecture. Different custom instructions can also share datapaths that share the same function, in a time-multiplexed way. It thereby reduces the reconfiguration time that would be needed to bind all the datapaths of a custom instruction.

Figure 6.2 – Example of binding three custom instructions using first-fit strategy with datapath reusing scheme



Source: (DINIZ, 2014).

In the first phase, datapaths from SAD custom instruction are bound to the first four FG-PRCs and two CGs. In the second phase, datapaths from SATD custom instruction are bound. The selected SATD implementation version in the example employs three Repack datapaths. As there are already two CG elements configured with Repack datapath, one more CG element is reconfigured and the other two could be reused by SATD. The other datapaths of SATD are bound to the first FG element (FG0). However, FG0 does not have enough free PRCs when binding the second Transform datapath. Then, the second Transform datapath is bound to the first PRC of the second FG element (FG1), i.e., the eighth PRC.

When considering the tile organization, binding the second Transform datapath to a different tile results in a high communication cost. It reduces performance when processing SATD instruction, compared to a situation where all datapaths are bound into the same tile. In the third phase, datapaths of DCT custom instruction are bound. All datapaths of DCT custom instruction can be reused by the already bound datapaths (in first and second phases), because they share the same function. However, the inter-tile communication remains, as the Transform datapath is bound to a different tile.

This first-fit with datapath reusing scheme prioritizes filling the first tile before binding to the second tile, reducing fragmentation and maintaining free contiguous elements, similar to other FG strategies (WALDER, 2003). However, it may bind

datapaths of one custom instruction to two different tiles, incurring in a high communication cost because of the interconnection between tiles. Such a high communication cost degrades not only the performance of an ill-bound custom instruction (e.g., SATD), but all the custom instructions that have some reused elements (e.g., DCT). In short an efficient binding scheme needs to account for the inter-tile communication costs to maximize the overall performance of the combined set of custom instructions.

6.3 Run-time Accelerator Binding Scheme

6.3.1 Problem Formulation

Custom instructions are employed to accelerate computational kernels of all concurrently executing tasks that receive a share of the reconfigurable fabric by the task allocation algorithm. The set of custom instructions to be accelerated at a given time is defined by $I=\{I_1,\ldots, I_n\}$. Each custom instruction I_i is composed of a set of datapaths $D(I_i)=\{D_1,\ldots, D_m\}$ in which m is the number of datapaths obtained after the custom instruction selection algorithm that also determines the granularity of each datapath, i.e. $g(D_k)=CG$ or $g(D_k)=FG$.

Our architecture is composed by a set of tiles $T=\{T_1,\ldots, T_l\}$ and a set of reconfigurable elements $E=\{E_1,\ldots,E_r\}$, such that its granularity is given as $g(E_{ej})=CG$ or $g(E_{ej})=FG$. Each element E_{ej} belongs to a certain tile T_j. All tiles have the same number of CG and FG elements. The function $t(E_{ej})$ returns the tile of element E_{ej}. Each tile is associated to a coordinate (x, y) that corresponds to the horizontal/vertical position of the tile in the floorplan. The total number of tiles |T|, the number of CG elements $n_{CG}(T_j)$ and PRCs FG elements $n_{FG}(T_j)$ inside a tile T_j are fixed at design-time. Two tiles directly connected through interconnection structure are able to communicate data from one datapath in c cycles. A binding algorithm is defined as a function $b : D \rightarrow E$ such that each datapath of custom instruction is mapped to a unique reconfigurable element. For instance, two datapaths (D_1 and D_2) of a custom instruction are mapped to the first two elements: $b(D_1)=E_1$, $b(D_2)=E_2$.

6.3.2 Run-time Accelerator Binding Scheme

The pseudo-code of our run-time accelerator binding scheme is shown in Figure 6.3. It receives the set of datapaths of a given selected implementation version of a custom instruction I_i (of a given task) to be bound. We assume that custom instructions from the multiple concurrently executing tasks are delivered sequentially to our scheme. Moreover, the architectural characteristics like number and performance properties of CG- and FG- reconfigurable elements and the tile structure are available to our scheme.

The first phase of our scheme is to choose the best tile (Section 6.3.3) to bind all or most of datapaths of a given custom instruction. As discussed earlier, it is beneficial that datapaths of the same custom instruction are mapped to reconfigurable elements in the same tile to achieve high performance by avoiding the frequent and high cost of inter-tile communication. In some case, it is possible to bind all datapaths of a custom instruction inside the best tile, i.e. there are enough free elements in the best tile that fit all the datapaths of a custom instruction (min_diff_DPs equal to 0). In this case, our scheme simply binds datapaths inside the best tile (Section 6.3.5). When not all datapaths of a custom instruction fit into the best tile, the remaining datapaths are bound into other tiles in a way to minimize the total communication cost with datapaths already bound into the best tile (Section 6.3.4).

6.3.3 Choosing the Best Tile to Bind a Custom Instruction

We propose a new tile-aware datapath reusing method to determine the so-called best tile to bind one custom instruction. We define the best tile as the tile in which all or at least most of the datapaths of a custom instruction must be bound. This method is shown in lines 2-16 of Figure 6.3. The first goal of our method is to bind all datapaths of one custom instruction onto the same tile. If there are no tiles with enough free elements available, the second goal is to find a tile where the minimum number of datapaths is bound in other tile. Before looking for free elements, we employ datapath reusing (line 4 in Figure 6.3) inside each tile, i.e. reusing datapaths from the custom instruction to be bound and the datapaths already bound in the current tile. When datapaths are reused across custom instructions, only the remaining datapaths (that cannot be reused) need to be bound. This results in fewer datapaths to be bound. This is beneficial for two reasons: 1) The method increases the probability of binding all datapaths of one custom instruction inside the same tile, reducing the inter-tile communication cost; 2) When fewer datapaths need to be reconfigured, less reconfiguration time is needed. After evaluating the reusing condition, if there are free elements available in the current tile, the method stops the search (break in line 15 in Figure 6.3) and associates the best tile as the current tile T_j. In this case, the algorithm employs first-fit algorithm inside the best tile (line 17 in Figure 6.3, see Section 6.3.5) and finishes.

Figure 6.3 – Pseudo-code of our run-time accelerator binding sheme.

```
Binding ( ) :
Inputs:          I_i: custom instruction I_i to be bound
       E : reconfigurable elements of the architecture.
       T: tile organization.
Output: b : D → E : mapping each datapath of I_i to one element.
begin
1. j = 0; min_diff_DPs = MAX_INT; best_tile = 0;
2. for each tile T_j {
3.    if ( nf(T_j) < n(T_j) ) {            // datapaths already bound in T_j
4.      (CGs_reused, PRCs_reused) = datapath_reusing(D(I_i),E);
5.   }
6.   else { CGs_reused = 0; PRCs_reused = 0; }
7.   n_CG_to_bind(I_i) = n_CG(I_i) – CGs_reused;
8.   n_FG_to_bind(I_i) = n_FG(I_i) – PRCs_reused;
9.   diff_DPs = Clip(n_CG_to_bind - nf_CG(T_j)) +
10.                Clip(n_FG_to_bind - nf_FG(T_j));
11.  if (diff_DPs > 0 ) {            // not all datapaths fit into tile T_j
12.       if (diff_DPs < min_diff_DPs) {
13.          min_diff_DPs = diff_DPs; best_tile = T_j }
14.   }
15.  else { min_diff_DPs = 0; best_tile = T_j; break; }
16. }
17. first_fit_binding(best_tile, n_CG_to_bind(I_i), n_FG_to_bind(I_i));
18. if ( min_diff_DPs > 0 ) {
19.    compute_comm_cost(T,best_tile);       // see Eq.1
20.    sort_by_increasing_comm_cost(T);
21.    for each tile T_j {
22.       if (n_CG_to_bind(I_i) > 0 or n_FG_to_bind(I_i) > 0) {
23.          datapath_reusing(D(I_i),E);
24.          first_fit_binding(T_j, n_CG_to_bind(I_i), n_FG_to_bind(I_i));
25.       }
26.       else return;
27.    }
28. }
29. else return;
end
```

Source: the author.

After the reusing procedure, if still there are no free elements, our method applies a greedy approach to determine the best tile (lines 11-14 in Figure 6.3). First, it computes the number of datapaths that cannot be bound to this tile (diff_DPs) using the number of CG- and FG- datapaths to bind (n_{CG}_to_bind and n_{FG}_to_bind) and the free CG- and FG- elements available to bind in tile T_j ($nf_{CG}(T_j)$ and $nf_{FG}(T_j)$). If the number of datapaths that cannot be bound to this tile is lower than the minimum number of datapaths that cannot be bound among all tiles evaluated (line 12 in Figure 6.3), it updates the minimum number min_diff_DPs and repeats all the procedure.

In the last case, the algorithm may loop over all tiles and may not find a tile with enough free elements in one tile to bind all DPs of a custom instruction. Then, our method returns the index of the tile with the minimum number of elements that cannot be bound on one tile (min_diff_DPs), determined by a greedy approach.

6.3.4 Binding into Tiles with Low Communication Cost

In some cases, the number of datapaths of a custom instruction to be bound cannot fit into only the best tile (as shown in Section 6.3.3) because there are no available/free reconfigurable elements. Hence, the remaining datapaths must be bound to other tile(s) in addition to the initial best selected tile. In order to determine the best tiles to bind these remaining datapaths, our scheme estimates the communication cost between the selected best tile and the other tiles of the architecture. We assume that the communication cost between every two neighboring tiles in the architecture is fixed and consumes c cycles to transfer data of one datapath. Because of that, the communication cost between the best tile b with coordinates (xb, yb) and other tile i with coordinates (xi, yi) can be estimated by calculating the Manhattan Distance (MD) of the tile coordinates, as shown in Equation 6.1.

$$comm\_cost = | x_b - x_i | + | y_b - y_i | \quad (6.1)$$

After the communication cost is estimated for all tiles that have free elements, the tiles are sorted w.r.t. the increasing communication cost (see lines 19-20 in Figure 6.3). Finally, the remaining datapaths are bound to the tiles from low to high communication cost, until the list of datapaths to be bound is empty. Before binding the remaining datapaths in free reconfigurable elements, our scheme also try to employ datapath reusing (line 23 of Figure 6.3). If no reusing is found, we finally employ first-fit binding to bind datapaths in the remaining tiles (line 24 of Figure 6.3, see also Section 6.3.5).

6.3.5 Binding Datapaths inside a Tile

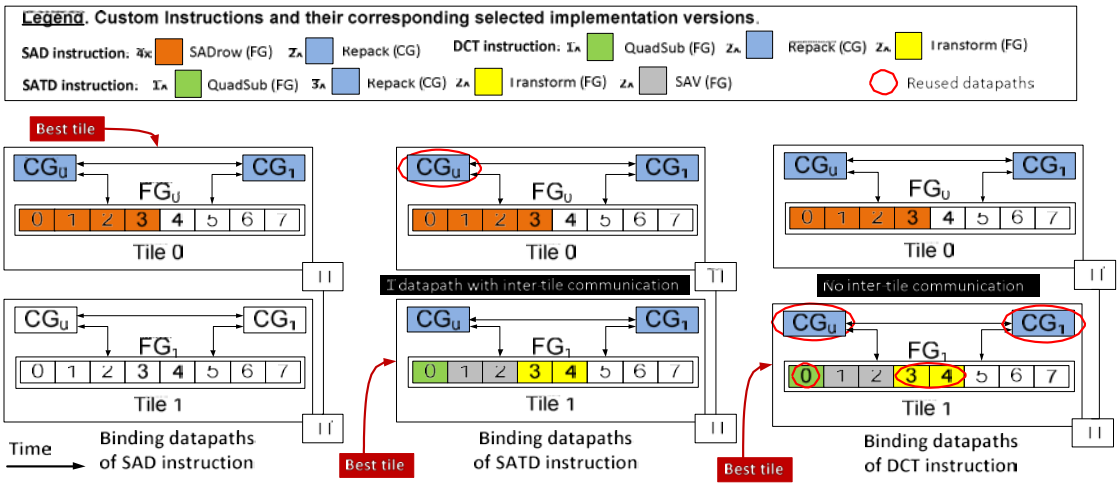
The binding problem of datapaths inside one tile is simplified to finding free CG- and FG- elements inside the selected tile. In this case, a simple first fit algorithm (first_fit_binding function, see lines 17 and 24 in Figure 6.3) provides a good binding solution inside a tile with reduced fragmentation problem. We assume that, inside a tile the communication cost between the reconfigurable elements is negligible due to a fast point-to-point interconnection between reconfigurable elements.

6.3.6 An Example of Our Binding Scheme

Figure 6.4 shows an example illustrating the procedure of our binding scheme using the same set of custom instructions and architecture as used in the motivational analysis (Section 6.2). Note in Figure 6.4 that, when binding the SATD custom instruction, our scheme selects tile 1 to bind most of datapaths (the so-called best tile). In this case, all

FG datapaths fit into tile 1 and only one Repack (CG) datapath has to be bound in other tile (tile 0). This remaining Repack (CG) datapath can be reused in tile 0. The number of datapaths communicating inter-tile in SATD custom instruction is 1 datapath using our scheme, instead of 2 datapaths when using first-fit strategy with datapath reusing scheme (see Section 6.2). When binding the DCT custom instruction, our scheme selects tile 1 as the best tile, because it can reuse all of its datapaths. No datapaths communicate inter-tile for DCT custom instruction when using our scheme, instead of 1 datapath when using first-fit strategy with datapath reusing scheme (see Section 6.2).

Figure 6.4 – Example of binding three custom instructions using our run-time accelerator binding scheme



Source: (DINIZ, 2014).

6.4 Results and Evaluation

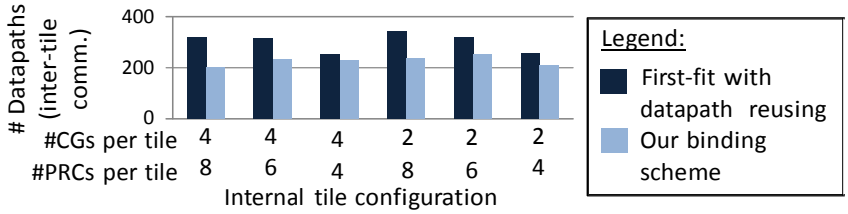
6.4.1 Experimental Setup

We considered a tile-based architecture with a fixed number of CG-elements and one FG-element with a fixed number of PRCs per tile. The number of CG and FG-PRC elements can be parameterized. Communication between two elements inside the same tile requires one cycle. Communication c between two elements in neighboring tiles (i.e., when MD = 1) requires 10 cycles. For the evaluation, we employed a H.264/AVC video encoder application and their design-time available custom instructions (SHAFIQUE, 2011) and the selection algorithm in (AHMED, 2011a). This application exhibits various compute-intensive kernels with both control- and data-flow dominant processing. Custom instruction selection is not aware of the tile structure, but it considers the total number of CG and FG elements in the architecture when selecting an appropriate implementation version for each custom instruction. The complete system is evaluated with KAHRISMA cycle-accurate instruction-set simulator (AHMED, 2011a). The inputs of the simulator (i.e., datapath latency for FG- and CG-fabrics) are obtained after place-and-route using Xilinx FPGA tools (i.e., Virtex-4) and ASIC-synthesis-flow for TSMC 90 nm (i.e., the same technology of FPGA). We considered a reconfiguration bandwidth of the FG-element of 67584 KB/s.

6.4.2 Evaluation of inter-tile communications

This section evaluates the efficiency of our scheme (compared to first-fit with datapath reusing; see motivational analysis in section 6.2) in determining the best tile for binding most of datapaths of a custom instruction. For simplicity, we show the results for architectures with only 2 tiles for 6 different combinations on the number of CG and FG elements per tile. Figure 6.4 shows the results of the number of datapaths communicating inter-tile, accumulated over all the bound custom instructions of the application. Our scheme reduces the number of inter-tile communicating datapaths by up to 37% (23% on average) compared to first-fit scheme. This result indicates that a tile-aware reusing can reduce the number of inter-tile communication compared to first-fit with datapath reusing scheme.

Figure 6.4 – Number of datapaths with inter-tile communication for 2 tiles



Source: (DINIZ, 2014).

6.4.3 Evaluation of communication overhead for many tiles

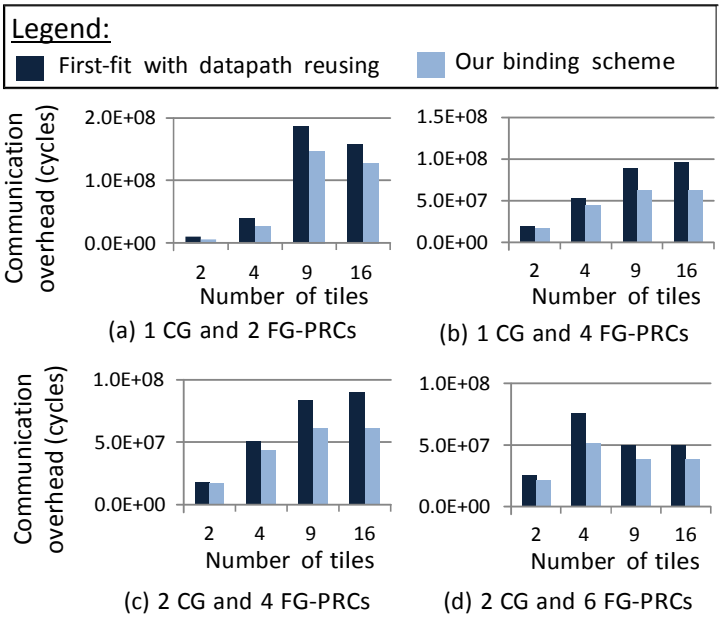
This section analyzes the communication overhead (in number of cycles) of our scheme compared to first-fit with datapath reusing scheme. For that, we have selected four different internal tile organizations with: 1) 1 CG and 2 FG-PRC elements; 2) 1 CG and 4 FG-PRC elements; 3) 2 CG and 4 FG-PRC elements; 4) 2 CG and 6 FG-PRC elements. We keep the tile size small (low number of CG and FG-PRCs) to better evaluate the efficiency of our method to bind datapaths into tiles with low communication cost. We show results for architectures with 2, 4 (i.e., 2x2), 9 (i.e., 3x3) and 16 (i.e., 4x4) tiles.

To compute the communication overhead of each bound custom instruction, we consider four parameters: 1) the number of datapaths bound to each tile; 2) the MD between the tiles that have datapaths of the custom instruction bound into it and the best tile; 3) the number of cycles c to communicate between two neighboring tiles; 4) the number of times the custom instruction is executed in the application. The results for all internal tile organizations are shown in Figure 6.5.

Our scheme reduces the communication overhead compared to first-fit scheme for all the configurations. The communication overhead is larger in the configurations with lower number of resources, because it is more probable that the custom instructions need more than one tile to be bound. Even for these cases, our scheme still benefits with lower communication cost. In summary, our scheme reduces the communication overhead by up to 44% (23% on average) compared to first-fit with datapath reusing scheme. This is achieved due to the combination of our two novel methods of tile-aware

datapath reusing and binding into tiles with low communication cost that integrate our run-time accelerator binding scheme.

Figure 6.5 – Communication overhead for different tile internal organizations



Source: (DINIZ, 2014).

7 CONCLUSIONS AND FUTURE WORK

The presented thesis focused on the contribution of novel dedicated and reconfigurable hardware accelerators for the HEVC Standard.

The research work started with an analysis of the HEVC encoding application, presented in Chapter 3, which indicates that the most important computations to be accelerated in hardware are those of the Interpolation Filter, of the Deblocking Filter, and of the calculation of the Sum of Absolute Differences needed for motion estimation. Our analysis shows that the results vary significantly depending on the video sequence (given as input by the user) and on the quantization parameter which in turn defines the level of video encoding loss being incurred. A run-time analysis on the Interpolation Filter coding tool indicates that there is a great potential of power/energy saving by adapting the hardware accelerator to the varying workload.

The results obtained of our novel dedicated hardware accelerators (Chapter 4) indicate significant gains over state-of-the-art hardware accelerators. The proposed dedicated hardware architecture for Interpolation Filter achieves sufficient throughput to process ultra-high resolution videos while reducing hardware area by ≈50% compared to a state-of-the-art interpolation architecture. It was achieved by designing area-efficient configurable multiplier-less datapaths. The throughput was improved through the use of two 12-pixel-parallel acceleration engines containing the configurable datapaths. A scheduling module was also designed to prevent pipeline stalls and to improve memory locality, reducing memory usage. The proposed dedicated hardware architecture for the Deblocking Filter achieves throughput similar with the state of the art with 5X to 6X reduction in gate count and 3X reduction in power dissipation. The datapaths developed in this work are highly optimized for area and employed hardware reuse. Our comparative analysis of SAD processing elements introduced various architecture design alternatives in order to explore different area, performance and power tradeoffs.

The Reconfigurable Interpolation Filter Hardware Architecture for HEVC standard, described in Chapter 5, is new and it provides significant design-time area reduction and run-time power/energy adaptation in a picture-by-picture basis. This feature was not yet supported by state-of-the-art interpolation filter architectures. Run-time adaptation is performed through a Prediction Scheme that estimates the number of interpolation filter calls and an Implementation Selection Version module that adapts to different throughput by selecting from a set of implementation versions.

Our novel Run-Time Accelerator Binding Scheme for tile-based mixed-grained reconfigurable architectures, presented in Chapter 6, reduces the communication overhead, compared to first-fit strategy with datapath reusing scheme, by up to 44% (23% on average) for different number of tiles and internal tile organizations. Our run-time accelerator binding scheme is aware of the underlying architecture to bind datapaths in an efficient way to avoid inter-tile communications.

The overall results demonstrate that the novel dedicated and reconfigurable hardware accelerators and techniques proposed in this thesis are in front of the state of the art solutions. Due to the power and energy limitations of current CMOS technologies and the high performance requirements of next-generation video coding standards, future video coding system implementations will integrate, in the same chip,

many-core processors with many dedicated and reconfigurable accelerators in a so-called accelerator-rich architectures. The accelerator-rich architectures are needed for high performance ultra-high resolution video encoding/decoding in real time with power/energy efficiency. In this context, this thesis introduces novel hardware accelerators and techniques which enable next-generation video coding standard implementations with improved area, performance, and power/energy efficiency.

7.1 Future Work

Beyond the novel contributions presented in this thesis work, several research directions arise for the future, which were not addressed in this work. Some of those research directions are suggested in this section as future work.

Hardware accelerators for other HEVC coding tools: while this thesis focused on the most compute-intensive kernels of HEVC standard, there are other coding tools not addressed that can be implemented as hardware accelerators to design a complete video coding system, such as Sum of Absolute Transform Differences (SATD), Context-Adaptive Binary Arithmetic Coding (CABAC), Intra Prediction, Transforms, Quantization, and Sample Adaptive Offset (SAO) Filter. The research challenge here is to design performance/area/power efficient hardware accelerators for these coding tools compared to the ones already presented in the literature.

Exploiting the HEVC parallel coding tools: as reviewed in Section 2.2, HEVC includes some parallel coding tools to facilitate parallel processing of video coding, such as Tiles and Wavefront Parallel Processing (WPP). Regarding tiles, there are many research challenges that affect both performance and video quality. The number of tiles for each picture and where the tile boundaries are placed is decided at video encoder side and it is not standardized. Breaking pictures into more tiles increase scalability for many-core processors, but decreases video quality. Hence, there is a tradeoff between performance and video quality when using tiles. Regarding WPP (where some CTUs of a picture can be processed in parallel in a multi-thread approach), the research challenge is to decide in which situations is beneficial to use it. There is also a decision on using tiles or WPP for each picture, because HEVC standard does not support both tools together yet.

Hardware accelerators for image and video processing: some methodologies are used in this thesis work to design efficient hardware accelerators for specific kernels of HEVC video coding standard. These methodologies may be applied to design hardware accelerators for other image and video applications, such as image processing (e.g. filtering, interpolation), image coding, video pre-processing, etc.

Accelerator-rich architectures for the Dark Silicon Era: the benefits of hardware accelerators compared to general purpose processors is pushing the multi-core and many-core processor research in the direction of accelerator-rich architectures, i.e. coupling the multi/many-core processors with many dedicated and reconfigurable hardware accelerators for specific kernels of applications. In this research topic, many research challenges arise. First of all, research challenges at design time need to be addressed. Good questions are what accelerators to design and whether they are designed as dedicated or reconfigurable kernels. Application profiling can help designers to choose what accelerators to design. Important kernels used in many applications and/or in successive application generations, are good candidates for dedicated accelerators. Reconfigurable accelerators can be used to provide flexibility for

important emerging kernels (e.g. the new Interpolation Filter of HEVC) and to map other accelerators that are used in only some phases of application execution (as reconfigurable accelerators are more power hungry). In the higher-level system architecture design, other research challenge is how to couple those many hardware accelerators to many processing cores, and how to connect them. Design time decisions are limited by chip area and thermal design power. Another research challenge appears for the run-time control: how to deal with the workload unbalance of threads in the accelerator-rich architecture. A run-time system is essential to allocate threads and power-on/off accelerators with the main goal to sustain performance under a power upper limit. The run-time system should be fed by application monitoring and prediction to improve the final application performance in power and timing.

7.2 Published Papers by the Author

This section presents a list of original research papers developed within the scope of this thesis. One journal paper and five conference papers were published during the development of the research which led to this thesis.

7.2.1 Journal Paper

DINIZ, C. M., SHAFIQUE, M., BAMPI, S., HENKEL, J. A Reconfigurable Hardware Architecture for Fractional Pixel Interpolation in High Efficiency Video Coding. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. v. 34, pp. 238-251, 2015.

7.2.2 Conference and Symposia Papers

DINIZ, C. M., FONSECA, M. B., COSTA, E. BAMPI, S. Enhancing a HEVC Interpolation Filter Hardware Architecture With Efficient Adder Compressors. In: 13th IEEE International NEW Circuits and Systems (NEWCAS) Conference, 2015, Grenoble.

DINIZ, C. M., SHAFIQUE, M., DALCIN, F., BAMPI, S. HENKEL, J. A Deblocking Filter Hardware Architecture for the High Efficiency Video Coding Standard. In: Design, Automation & Test in Europe Conference and Exhibition (DATE), 2015, Grenoble. pp. 1509-1514.

DINIZ, C. M., SHAFIQUE, M., BAMPI, S. HENKEL, J. Run-time accelerator binding for tile-based mixed-grained reconfigurable architectures. In: 24th International Conference on Field Programmable Logic and Applications (FPL), 2014, Munich. 4 p. 1-4.

DINIZ, C. M., SHAFIQUE, M., BAMPI, S. HENKEL, J. High-throughput interpolation hardware architecture with coarse-grained reconfigurable datapaths for HEVC. In: 20th IEEE International Conference on Image Processing (ICIP), 2013, Melbourne. pp. 2091-2095.

DINIZ, C. M., CORRÊA, G., AGOSTINI, L. V., SUSIN, A. A., BAMPI, S. Comparative Analysis of Parallel SAD Calculation Hardware Architectures for H.264/AVC Video Coding. In: IEEE Latin American Symposium on Circuits and Systems (LASCAS), 2010, Foz do Iguaçu. pp. 132-135.

REFERENCES

ABRAMOWSKI, A. and PASTUSZAK, G. A; Novel Intra Prediction Architecture for the Hardware HEVC Encoder. In: EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN (DSD), 2013...Proceedings [S.l.:s.n.], 2013, p. 429-436.

AFONSO, V., H. MAICH, L. AGOSTINI, D. FRANCO; Low cost and high throughput FME interpolation for the HEVC emerging video coding standard. In: IEEE LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS (LASCAS), 2013… Proceedings New York: IEEE, 2013, p.1-4

AGOSTINI, L. V., Desenvolvimento de Arquiteturas de Alta Performance Dedicadas à Compressão de Vídeo Segundo o Padrão H.264. Doctor Thesis - Universidade Federal do Rio Grande do Sul. Graduate Program in Computer Science, Porto Alegre, RS, 2007.

AHMADINIA, A. et al. Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices. IEEE Transactions on Computers, v. 56 n. 5, 2007, pp. 673–680.

AHMED, W. et al; Adaptive resource management for simultaneous multitasking in mixed-grained reconfigurable multi-core processors. In: INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESIGN AND SYSTEM SYNTHESIS (CODES+ISSS), 2011…Proceedings [S.l.]: ACM, 2011, p. 365–374.

AHMED, W., SHAFIQUE, M., BAUER, L., HENKEL, J; mRTS: Run-time system for reconfigurable processors with multi-grained instruction-set extensions. In: DESIGN, AUTOMATION & TEST IN EUROPE (DATE) Conference, 2011…Proceedings New York: IEEE, 2011, p. 1554–1559.

ALTERA Corporation. Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs. White Paper (WP-01137-1.0), 2010.

ALTERMANN, J. S., COSTA, E. A. C., BAMPI, S.; Fast forward and inverse transforms for the H.264/AVC standard using hierarchical adder compressors. In: 18TH IEEE/IFIP VLSI SYSTEM ON CHIP CONFERENCE (VLSI-SOC), 2010…Proceedings New York: IEEE, 2010, p. 310-315

BAUER, L. et al; Run-time instruction set selection in a transmutable embedded processor. In: DESIGN AUTOMATION CONFERENCE (DAC), 2008…Proceedings [S.l.]: ACM, 2008, p. 56–61.

BAUER, L., M. SHAFIQUE, J. HENKEL. Efficient Resource Utilization for an Extensible Processor Through Dynamic Instruction Set Adaptation. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.16, no.10, pp.1295-1308, 2008.

BAUER, L., M. SHAFIQUE, S. KREUTZ, J. HENKEL; Run-time system for an extensible embedded processor with dynamic instruction set. In: DESIGN, AUTOMATION & TEST IN EUROPE (DATE) CONFERENCE, 2008…Proceedings New York: IEEE, 2008, p. 752–757.

BECKER, T., LUK, W., CHEUNG, P. Y. K; Energy-Aware Optimisation for Run-Time Reconfiguration. In: IEEE ANNUAL INTERNATIONAL SYMPOSIUM ON FIELD-

PROGRAMMABLE CUSTOM COMPUTING MACHINES (FCCM), 2010… Proceedings [S.l.]: IEEE, 2010, p.55-62

BJONTEGAARD, G. Calculation of average PSNR differences between RD-curves. In: Video Coding Experts Group Meeting (document VCEG-M33), Austin, Apr. 2011.

BOSSEN, F. Common Test Conditions and Software Reference Configurations, Joint Collaborative Team on Video Coding Meeting (document JCTVC-L1100), Geneva, Jan. 2013.

BOSSEN, F., BROSS, B., SUHRING, K., FLYNN, D. HEVC Complexity and Implementation Analysis. IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp. 1685-1696

CADENCE Encounter RTL Compiler, http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx. Access: Nov. 2014

CHANG, S., CHENG, C.-C. CHEN, L.-G.; System Architecture Design Methodology for H.264/AVC Encoder. In: IEEE INTERNATIONAL SYMPOSIUM ON CONSUMER ELECTRONICS (ISCE), 2007…Proceedings New York: IEEE, 2007, p. 1-5

CHEN, L., MARCONI, T., MITRA, T.; Online scheduling for multi-core shared reconfigurable fabric. In: DESIGN, AUTOMATION & TEST IN EUROPE (DATE) CONFERENCE, 2012… Proceedings New York: IEEE, 2012, p. 582–585

CHEN, T.-C. et al., Analysis and Architecture Design of an HDTV720p 30 Frames/s H.264/AVC Encoder, IEEE Transactions on Circuits and Systems for Video Technology, Jun. 2006, pp. 673-688.

CHEN, T.-C., HUANG, Y.-W., CHEN, L.-G.; Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), 2004… Proceedings New York: IEEE, 2004, p. V-9-12.

CHUANG, T.-D. et al.; A 59.5mW scalable/multi-view video decoder chip for Quad/3D Full HDTV and video streaming applications. In: IEEE INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE (ISSCC), 2010…Proceedings New York: IEEE, 2010, p.330-331.

CISCO. Cisco Visual Networking Index: Forecast and Methodology, 2013–2018, White Paper, June 2014.

CLAUS, C. et al.; A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput. In: IEEE FIELD PROGRAMMABLE LOGIC AND APPLICATIONS (FPL), 2008…Proceedings New York: IEEE, 2008, p. 535–538.

COMPTON, K. and HAUCK, S. Reconfigurable computing: a survey of systems and software. ACM Computing Surveys, vol. 34, pp. 171-210, 2002.

CONCEICAO, R. et al.; Hardware design for the 32×32 IDCT of the HEVC video coding standard. In: 26TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2013…Proceedings [S.l.]: ACM, 2013, p.1-6

CONG, J. et al.; Accelerator-Rich Architectures: Opportunities and Progresses. In: DESIGN AUTOMATION CONFERENCE (DAC), 2014…Proceedings [S.l.]: ACM, 2014, p. 1-6

CORRÊA, G., et al., Performance and Computational Complexity Assessment of High-Efficiency Video Encoders. IEEE Transactions on Circuits and Systems for Video Technology, v. 22, n. 12, pp. 1899-1909, 2012.

COSTA, E. A. C., CORTES, F. P., CARDOZO, R., CARRO, L., BAMPI, S.; Modeling of Short Circuit Consumption Using Timing-Only Logic Cell Macromodels. In: 13TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2000…Proceedings [S.l.]: ACM, 2000, p. 222-227

DIAS, T., ROMA, N., SOUZA, L.; High performance multi-standard architecture for DCT computation in H.264/AVC High Profile and HEVC codecs. In: CONFERENCE ON DESIGN AND ARCHITECTURES FOR SIGNAL AND IMAGE PROCESSING (DASIP), 2013...Proceedings [S.l.:s.n.], 2013, p.14-21.

DING, L.-F. et al.; A 212MPixels/s 4096×2160p multiview video encoder chip for 3D/quad HDTV applications. In: IEEE INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE (ISSCC), 2009…Proceedings New York: IEEE, 2009, p.154-155

DINIZ, C. M., CORRÊA, G., AGOSTINI, L. V., SUSIN, A. A., BAMPI, S.; Comparative Analysis of Parallel SAD Calculation Hardware Architectures for H.264/AVC Video Coding. In: IEEE LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS (LASCAS), 2010…Proceedings New York: IEEE, p. 132-135

DINIZ, C. M., SHAFIQUE, M., BAMPI, S. HENKEL, J.; High-throughput interpolation hardware architecture with coarse-grained reconfigurable datapaths for HEVC. In: 20TH IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING (ICIP), 2013…Proceedings New York: IEEE, 2013, p. 2091-2095

DINIZ, C. M., SHAFIQUE, M., BAMPI, S. HENKEL, J.; Run-time accelerator binding for tile-based mixed-grained reconfigurable architectures. In: 24TH INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS (FPL), 2014…Proceedings New York: IEEE, 2014, p. 1-4.

DINIZ, C. M., SHAFIQUE, M., BAMPI, S., HENKEL, J. A Reconfigurable Hardware Architecture for Fractional Pixel Interpolation in High Efficiency Video Coding. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. v. 34, n.2, p. 238-251, Feb. 2015a.

DINIZ, C. M., SHAFIQUE, M., DALCIN, F., BAMPI, S. HENKEL, J.; A Deblocking Filter Hardware Architecture for the High Efficiency Video Coding Standard. In: DESIGN, AUTOMATION & TEST IN EUROPE (DATE) CONFERENCE, 2015…Proceedings New York: IEEE, 2015b, p. 1509-1514

DINIZ, C. M., FONSECA, M. B., COSTA, E., BAMPI, S.; Enhancing a HEVC Interpolation Filter Hardware Architecture With Efficient Adder Compressors. In: 13TH IEEE INTERNATIONAL NEW CIRCUITS AND SYSTEMS (NEWCAS) CONFERENCE, 2015…Proceedings New York: IEEE, 2015c

ESMAEILZADEH, H. et al.; Dark silicon and the end of multicore scaling. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA), 2011…Proceedings [S.l]: ACM, 2011, p. 365–376

FENLASON, J., R. Stallman. GNU gprof—The GNU Profiler, Free Software Foundation, Inc., 2000.

FORBES. Netflix Eyes Global Streaming Domination As It Crosses 50 Million Subscriber Mark. Available at: http://onforb.es/1ncjMod. Access: December, 2014.

FreePDK 45nm standard-cell library, Oklahoma State University, http://vlsiarch.ecen.okstate.edu/flow. Access: Nov. 2014

FRIEDMAN, S. et al.; SPR: An architecture-adaptive CGRA mapping tool. In: ACM/SIGDA INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS (FPGA), 2009…Proceedings [S.l]: ACM, 2009, p.191–200

GHANBARI, M. Standard Codecs: Image Compression to Advanced Video Coding. United Kingdom: The Institute of electrical Engineers, 2003.

GOOSSENS, K. et al.; Hardwired Networks on Chip in FPGAs to Unify Functional and Configuration Interconnects. INTERNATIONAL SYMPOSIUM ON NETWORKS-ON-CHIP (NOCS), 2008…Proceedings [S.l.:s.n.], 2008, p. 45-54.

GRUDNITSKY, A. et al.; Partial Online-Synthesis for Mixed-Grained Reconfigurable Architectures. In: DESIGN, AUTOMATION & TEST IN EUROPE (DATE) CONFERENCE, 2012…Proceedings New York: IEEE, 2012, p. 1555–1560

GUO, Z., ZHOU, D., GOTO, S.; An optimized MC interpolation architecture for HEVC. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), 2012…Proceedings New York: IEEE, p.1117-1120

HAMMED, R. et al., Understanding sources of inefficiency in general-purpose chips. Communications of the ACM, v. 54, n. 10, Oct 2011, pp. 85-93

HARTENSTEIN, R.; A Decade of Reconfigurable Computing: a Visionary Retrospective. Embedded Tutorial. In: DESIGN AUTOMATION CONFERENCE (DAC), 2001…Proceedings [S.l]: ACM, 2001, p. 642-649

HEVC Test Model (HM) version 10.0, 2013. http://hevc.hhi.fraunhofer.de.

HUANG, Y.-W. et al. Survey on Block Matching Motion Estimation Algorithms and Architectures with New Results. Journal of VLSI signal processing systems for signal, image and video technology, Springer, v. 42, n.3, 2006, pp. 297-320

IAN, K. and ROSE, J., Measuring the Gap Between FPGAs and ASICs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 26, n. 2, pp. 203–215, 2007

ISO, Coding of Audio-Visual Objects—Part 2: Visual, ISO/IEC 14496-2 (MPEG-4 Visual version 1), ISO/IEC JTC 1, Apr. 1999.

ISO, Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s—Part 2: Video, ISO/IEC 11172-2 (MPEG-1), ISO/IEC JTC 1, 1993.

ITU-T and ISO/IEC JTC 1, Advanced video coding for generic audiovisual services, ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), 2003.

ITU-T and ISO/IEC JTC 1, Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video, ITU-T Rec. H.262 and ISO/IEC 13818-2 (MPEG 2 Video), Nov. 1994.

ITU-T and ISO/IEC, High Efficiency Video Coding, ITU-T Recommendation H.265 and ISO/IEC 23008-2, 2013.

ITU-T, Video Codec for Audiovisual Services at 64 kbit/s, ITU-T Rec. H.261, version 1: Nov. 1990.

ITU-T, Video Coding for Low Bit Rate Communication, ITU-T Rec. H.263, Nov. 1995.

IYER, R. et al.; Accelerator-rich architectures: Implications, opportunities and challenges. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE (ASP-DAC), 2012…Proceedings [S.l.:s.n.], 2012 p.106-107

JAFRI, S. et al.; Compact generic intermediate representation (CGIR) to enable late binding in coarse grained reconfigurable architectures. In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE TECHNOLOGY (FPT), 2011…Proceedings [S.l.:s.n.], 2011, p. 1–6

KAO, C.-Y., WU, C.-L., LIN, Y.-L. A High-Performance Three-Engine Architecture for H.264/AVC Fractional Motion Estimation. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v. 18, n. 4, pp.662-666, 2010

KHAN, M. U. K., SHAFIQUE, M., GRELLERT, M., HENKEL, J.; Hardware-software collaborative complexity reduction scheme for the emerging HEVC intra encoder. In: DESIGN, AUTOMATION & TEST IN EUROPE (DATE) CONFERENCE, 2013…Proceedings New York: IEEE, 2013, p.125-128

KIM, I.-K. MCCANN, SUGIMOTO, K. BROSS, B. HAN, W.-J. HM10: High Efficieny Video Coding (HEVC) Test Model 10 Encoder Description, JCT-VC-L1002, JCT-VC, Geneva, Jan. 2013.

KIM, I.-K., K. McCann, K. Sugimoto, B. Bross, W.-J. Han. HM10: High Efficieny Video Coding (HEVC) Test Model 10 Encoder Description, JCT-VC-L1002, JCT-VC, Geneva, Jan. 2013.

KIM, I.-K., K. McCann, K. Sugimoto, B. Bross, W.-J. Han. HM7: High Efficieny Video Coding (HEVC) Test Model 7 Encoder Description, JCT-VC-I1002, JCT-VC, Geneva, Apr-Mai. 2012.

KIM, I.-K., K. McCann, K. Sugimoto, B. Bross, W.-J. Han. HM8: High Efficieny Video Coding (HEVC) Test Model 8 Encoder Description, JCT-VC-J1002, JCT-VC, Stockholm, Jul. 2012.

KLEIN, M. White Paper WP298, Power consumption at 40 and 45 nm. Xilinx, 2009.

KOKARAM, A. Challenges in Video Ingest at YouTube. Special Panel on Large Scale Image and Video Repositories. In: PICTURE CODING SYMPOSIUM (PCS), 2013…Proceedings [S.l.:s.n.], 2013

KÖNIG, R. et al.; KAHRISMA: A novel hypermorphic reconfigurable instruction-set multi-grained-array architecture. In: DESIGN, AUTOMATION & TEST IN EUROPE (DATE) CONFERENCE, 2010…Proceedings New York, IEEE, 2013, p. 819–824.

LI, Y. and Y. He.; Bandwidth Optimized and High Performance Interpolation Architecture in Motion Compensation for H.264/AVC HDTV Decoder. Journal of Signal Processing Systems, v. 52, pp. 111–126, 2008.

LI, Z., HAUCK, S.; Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation. In: ACM/SIGDA INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS (FPGA), 2002…Proceedings [S.l.]: ACM, 2002, p. 187-195.

LIU, Z., et al.; 32-Parallel SAD Tree Hardwired Engine for Variable Block Size Motion Estimation in HDTV1080P Real-Time Encoding Application. In: IEEE SYMPOSIUM ON SIGNAL PROCESSING (SIPS), 2007…Proceedings New York: IEEE, 2007, p. 675-680.

LU, L., MCCANNY, J. V., SEZER, S. Subpixel Interpolation Architecture for Multistandard Video Motion Estimation. IEEE Transactions on Circuits and Systems for Video Technology, v.19, n.12, pp.1897-1901, 2009

LV, H., WANG, R., XIE, X., JIA, H., GAO, W. A comparison of fractional-pel interpolation filters in HEVC and H.264/AVC. In: IEEE Visual Communications and Image Processing (VCIP), 2012, pp.1-6, 2012.

MCCANN, K. et al., HM6: High Efficieny Video Coding (HEVC) Test Model 6 Encoder Description", JCT-VC-H1002, JCT-VC, San Jose, Feb. 2012.

MOORE, G. "Cramming more components onto integrated circuits", Electronics Magazine, 1965, p. 4

NALLURI, P., ALVES, L. N., NAVARRO, A.; A novel SAD architecture for variable block size motion estimation in HEVC video coding. In: INTERNATIONAL SYMPOSIUM ON SYSTEM ON CHIP (SOC), 2013…Proceedings [S.l.:s.n.], 2013, pp.1-4

NANGATE, NanGate 45 nm Open Cell Library, www.nangate.com/?page_id=22. Access: Feb. 2015.

NORKIN, A., et al. HEVC Deblocking Filter. IEEE Transactions on Circuits and Systems for Video Technology, v. 22, n. 12, pp.1746-1754, Dec. 2012

OHM, J.-R., et al. Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC). IEEE Transactions on Circuits and Systems for Video Technology, v.22, n.12, pp.1669-1684, Dec. 2012

OZCAN, E., ADIBELLI, Y., HAMZAOGLU, I.; A high performance deblocking filter hardware for High Efficiency Video Coding. In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS (FPL), 2013…Proceedings New York, IEEE, 2013, p.1-4

PASTUSZAK, G. and JAKUBOWSKI, M. Adaptive Computationally Scalable Motion Estimation for the Hardware H.264/AVC Encoder. IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 5, pp. 802-812, May 2013.

PENG, B., et al.; A hardware CABAC encoder for HEVC. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2013…Proceedings New York: IEEE, 2013, p.1372-1375

PORTO, R. Desenvolvimento Arquitetural para Estimação de Movimento de Blocos de Tamanhos Variáveis Segundo o Padrão H.264/AVC de Compressão de Vídeo Digital, 2008. 96f. Master Dissertation (Master in Computer Science) – Instituto de Informática, UFRGS, Porto Alegre, Brazil.

PowerPlay Early Power Estimator for Stratix III/IV/V Devices, Altera, www.altera.com, 2014.

RICHARDSON, I. The H.264/AVC Advanced Video Compression Standard. 2nd Edition. Chichester: John Wiley and Sons, 2010.

ROUHOLAMINI, et al.; A New Design for 7:2 Compressors. In: IEEE/ACS INTERNATIONAL CONFERENCE ON COMPUTER SYSTEMS AND APPLICATIONS (AICCSA), 2007…Proceedings [S.l.:s.n.], 2007

SHAFIQUE, M. and J. Henkel. Hardware/software architectures for low-power embedded multimedia systems. Springer Science+Business Media, LLC, 2011.

SHAFIQUE, M. et al. Minority-Game-based resource allocation for run-time reconfigurable multi-core processors. In: DESIGN, AUTOMATION & TEST IN EUROPE (DATE) CONFERENCE, 2011…Proceedings New York: IEEE, 2011, pp. 1–6

SHAFIQUE, M., BAUER, L., HENKEL, J. Optimized Application Architecture of the H.264 Video Encoder for Application Specific Platforms. In: IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMEDIA), pp. 119-124, 2007.

SHAFIQUE, M., BAUER, L., HENKEL, J. Optimizing the H.264/AVC Video Encoder Application Structure for Reconfigurable and Application-Specific Platforms. Journal of Signal Processing Systems, vol. 60, no. 2, pp. 183-210, 2010.

SHAFIQUE, M., BAUER, L., HENKEL, J.; Selective Instruction Set Muting for Energy-Aware Adaptive Processors. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD), 2010…Proceedings [S.l]: ACM, 2010, p. 353-360

SHAFIQUE, M., BAUER, L., HENKEL, J.; REMiS: Run-time Energy Minimization Scheme in a Reconfigurable Processor with Dynamic Power-Gated Instruction Set. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD), 2009…Proceedings [S.l]: ACM, 2009, p. 55-62

SHAFIQUE, M., BAUER, L., HENKEL, J. Adaptive energy management for dynamically reconfigurable processors." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 33. n. 1,p. 50-63, Jan. 2014.

SHANG, L., KAVIANI, A. S., BATHALA, K.; Dynamic power consumption in Virtex-II FPGA family. In: ACM/SIGDA INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS (FPGA), 2002…Proceedings [S.l]: ACM, 2002, p. 157–164

SHEN, W., SHANG, Q., SHEN, S., FAN, Y., ZENG, X.; A high-throughput VLSI architecture for deblocking filter in HEVC. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2013…Proceedings New York: IEEE, 2013, p. 673-676

SHEN, S., et al; A pipelined VLSI architecture for Sample Adaptive Offset (SAO) filter and deblocking filter of HEVC. IEICE Electronics Express, v.10, n.11, pp. 1–11, 2013a

SMIT, G., et al.; Overview of the 4S project. In: IEEE SYSTEMS ON CHIP CONFERENCE (SOCC), 2005…Proceedings [S.l.:s.n.], 2005, p. 70–73

STANGHERLIN, K. H., BAMPI, S.; Energy-speed exploration for very-wide range of dynamic V-F scaling. In: 26TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2013…Proceedings [S.l.]: ACM, 2013, p. 1-6

SULLIVAN, G. J., et al., Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on Circuits and Systems for Video Technology, v. 22, n. 12, pp.1649-1668, Dec. 2012

SYNOPSYS, Synopsis PowerCompiler, datasheet available at homepage: http://www.synopsys.com/products/power/power_ds.pdf

TAYLOR, M.B. et al. A Landscape of the New Dark Silicon Design Regime, IEEE Micro, vol.33, no.5, pp.8,19, Sept.-Oct. 2013

THOMA, F. et al.; MORPHEUS: Heterogeneous reconfigurable computing. In: FIELD PROGRAMMABLE LOGIC CONFERENCE (FPL), 2007…Proceedings New York: IEEE, 2007, p. 409–414

TSUNG, P.-K., et al.; A 216fps 4096×2160p 3DTV set-top box SoC for free-viewpoint 3DTV applications. In: IEEE INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE (ISSCC), 2011…Proceedings New York: IEEE, 2011, p. 124-126

TSUNG, P.-K., et al.; Single-iteration full-search fractional motion estimation for quad full HD H.264/AVC encoding. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO (ICME), 2009…Proceedings New York: IEEE, 2009, p. 9-12

TUAN, T. et al.; A 90nm low-power FPGA for battery-powered applications. In: ACM/SIGDA INTERNATIONAL SYMPOSIUM ON FIELD PROGRAMMABLE GATE ARRAYS (FPGA), 2006…Proceedings [S.l]: ACM, 2006, p. 3-11

VANNE, J. et al., A High-Performance Sum of Absolute Difference Implementation for Motion Estimation, IEEE Transactions on Circuits and Systems for Video Technology, v. 16, n. 7, pp. 876-883, Jul. 2006

VANNE, J. et al., Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs, IEEE Transactions on Circuits and Systems for Video Technology, v. 22, n. 12, pp.1885-1898, Dec. 2012.

VENKATESH, G. et al.; Conservation cores: Reducing the energy of mature computations. In: International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2010…Proceedings [S.l]: ACM, 2010, p. 205-218.

WALDER, H., STEIGER, C., PLATZNER, M.; Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing. In: IEEE INTERNATIONAL PARALLEL & DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS), 2003…Proceedings [S.l.:s.n.], 2003, p 1-8

WANG, R. et al. High throughput and low memory access sub-pixel interpolation architecture for H.264/AVC HDTV decoder, IEEE Transactions on Consumer Electronics, v. 51, n. 3, pp. 1006-1013, 2005.

WANG, S.-Z. et al.; A new motion compensation design for H.264/AVC decoder. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2005…Proceedings New York: IEEE, 2005, p.4558-4561

WANG, Y.-J., CHENG, C.-C., CHANG, T.-S. A Fast Algorithm and Its VLSI Architecture for Fractional Motion Estimation for H.264/MPEG-4 AVC Video Coding. IEEE Transactions on Circuits and Systems for Video Technology, v. 17, n. 5, pp. 578-583, 2007.

WATKINS, M. and ALBONESI, D.; ReMAP: A reconfigurable heterogeneous multicore architecture. In: INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE (MICRO)…Proceedings New York: IEEE, 2010, p. 497–508

WEINBERGER, A. 4-2 carry-save adder module. [S.l.]: IBM, 1981. Technical Disclosure Bulletin.

WIEGAND, T. et al. Overview of the H. 264/AVC video coding standard. IEEE Transactions on Circuits and Systems for Video Technology, v. 13 n. 7, pp. 560-576, Jul. 2003

WIEGAND, T. et al. Rate-Constrained Coder Control and Comparison of Video Coding Standards. IEEE Transactions on Circuits and Systems for Video Technology, v. 13 n. 7, pp. 688-703, Jul. 2003

XILINX, Inc. "Partial Reconfiguration on Xilinx FPGAs", WP374, 2010.

XILINX, Inc. Virtex-5 FPGA Configuration User Guide, UG191 (v3.11), Oct. 2012.

XILINX, Inc. Xilinx Power Estimator, UG440 (v2013.2/14.6), Jun. 2013

XILINX, Xilinx ISE Design Suite version 14.4, http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm. Access: Nov. 2014

XILINX, Xilinx Virtex-6 Family Overview, DS150 (v2.4) http://www.xilinx.com/support/documentation/data_sheets/ds150, Jan. 2012

YI, L. et al.; A Communication Aware Online Task Scheduling Algorithm for FPGA-Based Partially Reconfigurable Systems. In: IEEE INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES (FCCM), 2010)…Proceedings New York: IEEE, 2010, p. 65–68

ZATT, B., et al. A Reduced Memory Bandwidth and High Throughput HDTV Motion Compensation Decoder for H.264/AVC High 4:2:2 Profile. Journal of Real-Time Image Processing, v. 8, n.1, pp. 127-140, Mar. 2013.

ZHOU, D. et al.; A 2Gpixel/s H.264/AVC HP/MVC video decoder chip for Super Hi-Vision and 3DTV/FTV applications. In: IEEE INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE (ISSCC), 2012…Proceedings New York: IEEE, 2012, p. 224-226

APPENDIX A <EXTENDED ABSTRACT IN PORTUGUESE>

A.1. Introdução

Hoje em dia, existem muitos dispositivos no mercado capazes de gravar e exibir vídeo digital, como televisores digitais smart, computadores de mesa e portáteis, tablets, smartphones, aparelhos de videogame, câmeras gravadoras, câmeras de segurança, etc. Estes dispositivos possibilitam uma variedade de aplicações de vídeo digital, como streaming de vídeo, transmissão de televisão digital, vídeo-conferência, cinema digital, vídeo-vigilância, etc. Dois serviços de vídeo digital sob demanda pela Internet, YouTube e Netflix, tornaram-se incrivelmente populares nos últimos anos. O Youtube é o maior repositório de vídeo e serviço de transmissão de vídeo pela Internet, com 80 horas de vídeo enviadas por minuto pelos usuários e milhões de visualizações por dia (KOKARAM, 2013). O Netflix é um serviço pago de transmissão de vídeo sob-demanda que contém vídeos e séries de TV. O Netflix atingiu a marca de 50 milhões de assinantes no segundo quadrimestre de 2014 (FORBES, 2014). É previsto que o tráfego de vídeo pela Internet será de 79% de todo o tráfego da Internet até 2018 (CISCO, 2014).

Para lidar com o armazenamento e transmissão de vídeo pela Internet (e por outras redes de comunicação), compressão de vídeo é essencial. A seguir, segue um exemplo do motivo pelo qual a compressão de vídeo é importante: um vídeo cru (não comprimido) de 10 minutos com resolução de 720x480 pixels (Standard Definition - SD) representado com 24 bits por pixel (8 bit para cada canal de cor, usando 3 canais de cor) e com 30 quadros por segundo (frames per second - fps) necessita de 19 gigabytes (GB) para ser armazenado ou transmitido pela Internet. O mesmo vídeo não comprimido de 10 minutos com resolução de 1920x1080 pixels (Full-HD) necessita de 112 GB. O mesmo vídeo no novo formato de resolução Sony 4K (4096x2160 pixels), usado na Copa do Mundo da FIFA de 2014, necessita de 477 GB. Não é viável lidar com este volume de dados de sequências de vídeo não comprimidas usando as tecnologias recentes de armazenamento e comunicação.

Codificação de vídeo é o processo de comprimir e decomprimir vídeo digital. Em outras palavras, codificação de vídeo é o processo de conversão de vídeo digital em um formato adequado para transmissão ou armazenamento. O número de bits para representar o vídeo codificado (comprimido) é reduzido comparado ao vídeo não codificado (não comprimido). A codificação de vídeo é baseada em um par complementar de sistemas: um codificador (compressor) e um decodificador (descompressor). O codificador de vídeo converte o vídeo não comprimido em uma forma comprimido, antes do armazenamento ou transmissão. Este processo é conhecido como codificação de vídeo. O decodificador de vídeo converte o vídeo da forma comprimida de volta à representação do vídeo original (ou muito similar ao original). Este processo também é conhecido como decodificação de vídeo. O par codificador/decodificador é muitas vezes descrito como CODEC (do inglês, enCOder/DECoder). A compressão de vídeo é feita pela remoção de redundâncias, ou seja, informações não necessárias para representação do vídeo. A compressão de vídeo também pode introduzir redundância subjetiva, ou seja, informação que pode ser removida sem que afete de forma significativa a percepção do observador da qualidade do vídeo. Se o vídeo decodificado é idêntico ao vídeo original não comprimido, o

processo de codificação é sem perdas. Na compressão com perdas, a redundância subjetiva também é aplicada, resultando em uma diferença entre o vídeo não comprimido e o vídeo decodificado (após ter sido comprimido). Compressão com perdas é aplicada para atingir maior compressão. A alta compressão resulta porém em uma redução da qualidade do vídeo decodificado comparado com o vídeo original não comprimido (RICHARDSON, 2010).

Os padrões de codificação de vídeo são desenvolvidos para codificar (comprimir) vídeos. A maioria dos padrões de codificação de vídeo aplica compressão com perdas para alcançar alta eficiência de compressão de vídeo. Quando os padrões de codificação de vídeo estão sendo desenvolvidos, o objetivo é comprimir vídeo com o mínimo de perda de qualidade para um certo tamanho do vídeo comprimido (ou para atingir o menor tamanho de vídeo comprimido para um certo alvo de qualidade de vídeo). Os padrões de codificação de vídeo evoluíram nas últimas duas décadas, principalmente impulsionados por novas aplicações de vídeo e o aumento na resolução dos vídeos. Os avanços nos padrões de codificação de vídeo recentes para prover aumento da eficiência de compressão de vídeo resultam em um imenso esforço computacional. É exigido dos dispositivos eletrônicos capazes de processar vídeo que proporcionem maior desempenho a cada geração de padrão de codificação de vídeo, para codificar e decodificar vídeos de alta resolução em tempo real. Neste contexto, a seção A.1.1 apresenta a motivação e a definição do problema que norteia esta tese.

A.1.1. Motivação e Definição do Problema

A recente demanda por vídeos de resolução ultra-alta (além de 1920x1080 pixels) impulsiona o desenvolvimento de novos padrões de codificação de vídeo mais eficientes para prover alta eficiência de compressão O mais novo e mais eficiente padrão de codificação de vídeo é o padrão High Efficiency Video Coding (HEVC), desenvolvido pelo Joint Collaborative Team on Video Coding (JCT-VC), formado por especialistas do Video Coding Experts Group (VCEG) da International Telecommunication Union (ITU) e do Motion Picture Experts Group (MPEG) da International Standardization Union (ISO). O HEVC foi publicado em Abril de 2013 como uma recomendação chamada ITU-T H.265 (ITU-T, 2013).

O HEVC atinge o dobro da eficiência de compressão (ou 50% de redução na taxa de bits) comparado com o padrão de codificação de vídeo mais eficiente até o momento, e mais utilizado no mercado, o padrão H.264/AVC (Advanced Video Coding) (ITU-T, 2011). O dobro da eficiência de compressão do HEVC sobre o H.264/AVC é atingido para qualidade de vídeo similar, pois ambos os padrões aplicam compressão com perdas. O HEVC atinge tal eficiência de compressão pelo uso de tamanhos de bloco maiores (para lidar com as resoluções maiores), particionamento de bloco sofisticado, e novos ferramentas de codificação avançadas (SULLIVAN, 2012).

A alta eficiência de compressão do HEVC resulta em um aumento do esforço computacional do codificador HEVC que varia de 1,2 a 3,2 vezes o esforço computacional do codificador H.264/AVC (VANNE, 2012). Isto requer melhoria de desempenho adicional dos dispositivos capazes de processar vídeo, para lidar com o aumento da complexidade e mesmo assim ser capaz de codificar vídeos de alta resolução em tempo real. Um esforço de pesquisa substancial, especialmente no codificador HEVC, é previsto para atingir este objetivo (BOSSEN, 2012).

No passado, a melhoria de desempenho era atingida devido aos avanços da tecnologia de fabricação dos semicondutores, que proporcionam altas frequências de operação através de transistores menores e mais rápidos. Recentemente, o avanço na fabricação em silício ainda proporciona transistores menores e mais rápidos a cada novo nodo tecnológico CMOS. Os chips continuam integrando mais transistores na mesma área, seguindo a lei de Moore para densidade (MOORE, 1965). As tecnologias CMOS recentes são capazes de integrar mais e mais núcleos (cores) de processamento no mesmo chip, chamados de processadores multi-core e many-core.

Entretanto, nas novas tecnologias abaixo de 100 nm, o desempenho é limitado por uma potência máxima denominada Potência Térmica de Projeto (Thermal Design Power), dado que atualmente a densidade de potência do transistor está aumentando a cada novo nodo tecnológico CMOS (ESMAEILZADEH, 2011). Para garantir que os chips permaneçam abaixo da potência térmica de projeto, nem todos os transistores do chip podem operar na máxima velocidade a todo o tempo, resultando no chamado "muro de utilização" (utilization wall) (VENKATESH, 2010). A região do chip que fica a maior parte do tempo operando a baixa frequência ou desligada é chamada em geral de "dark silicon" (ESMAEILZADEH, 2011) (TAYLOR, 2013). Um trabalho recente (TAYLOR, 2013) prevê que a porcentagem de dark silicon irá aumentar a cada nodo tecnológico e estará por volta de 94% até o ano 2020. Dark silicon irá limitar o aumento da frequência de operação dos processadores para proporcionar aumento de desempenho.

Para lidar com o alto desempenho necessário para o novo padrão HEVC e para manter os chips abaixo da potência térmica de projeto, os processadores no futuro irão integrar muitos aceleradores de hardware no chip para cada função específica da computação, chamados de arquiteturas ricas em aceleradores (IYER, 2012) (CONG, 2014). Aceleradores de hardware especializados são 500X mais eficientes em energia que processadores de propósito geral realizando a mesma função (HAMMED, 2011). Como as funções da computação não são executadas todo tempo simultaneamente, os aceleradores podem ser desligados quando não estão em uso. Logo, aceleradores de hardware especializados para funções intensivas em computação são uma maneira eficiente de "preencher" a área "escura" dos chips.

Apesar dos aceleradores de hardware dedicados proporcionarem alto desempenho e eficiência energética para codificação e decodificação de vídeo em tempo real, eles tem algumas desvantagens. Eles são fixos em tempo de projeto e não podem alterar o hardware em campo, depois da fabricação do chip em silício. Eles também possuem altos custos para o projeto e fabricação em silício. Hardware reconfigurável fornece uma solução com baixos custos de projeto, rápida chegada ao mercado, e flexibilidade para rápidas alterações do circuito através de reconfigurações dinâmicas (TUAN, 2006). Projetos baseados em Field-programmable Gate Array (FPGA) combinam o alto desempenho de aceleradores dedicados com a capacidade de explorar alto grau de paralelismo com alto grau de flexibilidade através de capacidade de programação e reconfiguração do hardware (SHAFIQUE, 2009)(COMPTON, 2002). A desvantagem dos FPGAs é sua alta potência comparada a aceleradores dedicados.

O autor desta tese prevê que tanto aceleradores dedicados como aceleradores reconfiguráveis serão usados nas futuras arquiteturas de processadores ricas em aceleradores. Esta tese apresenta diferentes contribuições, tanto em aceleração dedicada como reconfigurável, como é discutido na seção A.1.2. Estas contribuições podem ser aplicadas aos codificadores HEVC atuais e nos codificadores de vídeo de gerações

futuras, se estes codificadores forem baseados em unidades de codificação baseadas em blocos de pixels de vídeo.

A.1.2. Contribuições desta Tese

O objetivo desta tese é pesquisar aceleradores de hardware dedicados e reconfiguráveis inovadores para ferramentas de codificação mais intensivas em computação do novo padrão HEVC. Uma análise da aplicação HEVC, apresentada na seção A.2, verifica que as ferramentas mais importantes em termos de esforço computacional são o Filtro de Interpolação de Pixel Fracionário, o Filtro de Deblocagem e a computação da Soma das Diferenças Absolutas.

No domínio dos aceleradores dedicados, apresentados na seção A.3, esta tese introduz as seguintes contribuições inovadoras:

- Uma arquitetura de hardware de alta taxa de processamento para o Filtro de Interpolação de Pixel Fracionário do HEVC, que inclui núcleos de aceleração de luminância e crominância com 12 unidades de módulos de filtragem cada um.
- Uma arquitetura de hardware para o Filtro de Interpolação de Pixel Fracionário do HEVC utilizando somadores compressores para otimizar área, desempenho e dissipação de potência.
- Uma arquitetura de hardware de alta taxa de processamento para o Filtro de Deblocagem utilizando reuso de dados para acelerar a decisão do filtro com baixo custo de área.
- Uma análise comparativa de diferentes alternativas arquiteturais para o cálculo da Soma das Diferenças Absolutas em termos de área de hardware, taxa de processamento e dissipação de potência.

Com relação a aceleradores de vídeo reconfiguráveis, esta tese introduz adicionalmente as seguintes contribuições:

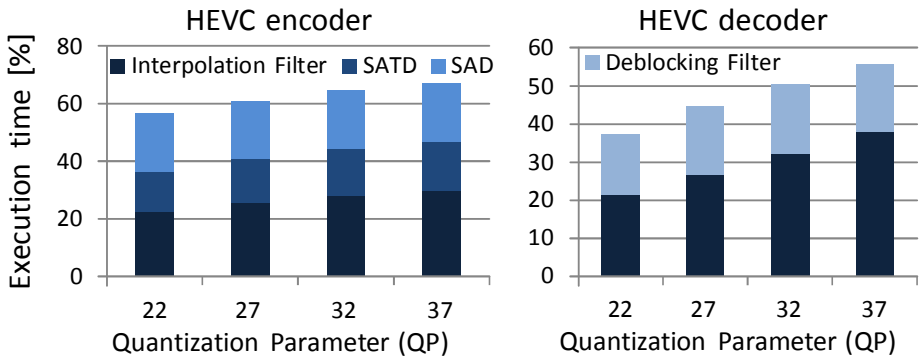
- Uma arquitetura reconfigurável para o Filtro de Interpolação de Pixel Fracionário do HEVC é apresentada na seção A.4. A arquitetura incorpora um esquema de predição para estimar o número de chamadas do filtro de interpolação a cada imagem, utilizando o conhecimento da estrutura de codificação. Um conjunto de diferentes versões de implementação para os núcleos de aceleração do filtro de interpolação são desenvolvidos para permitir um compromisso de área versus desempenho. Um esquema de seleção de versão de implementação é proposto. Ele seleciona uma versão de implementação do núcleo de aceleração para cada imagem, dependendo do número predito de chamadas do filtro de interpolação calculado pelo esquema de predição. Um esquema de escalonamento é introduzido para determinar a ordem do processamento e configurar os tipos de filtro. Ele facilita o reuso das entradas e previne leitura redundante das amostras de entrada.
- Um esquema de alocação de aceleradores em tempo de execução para arquiteturas reconfiguráveis de grão misto baseadas em tiles de processamento é apresentado na seção A.5. O esquema aplica reuso de módulos de hardware e estimativa de custo de comunicação entre tiles para calcular em tempo de execução uma alocação de aceleradores que minimize a comunicação entre tiles destes módulos.

A.2. Análise da Aplicação HEVC

A primeira etapa deste trabalho foi a análise da aplicação HEVC (codificador e decodificador). Esta análise suporta as decisões de quais aceleradores projetar e também algumas decisões arquiteturais. A análise se baseia no perfilhamento (profiling) do software do HEVC, o HM (HM, 2013). A contribuição de cada ferramenta de codificação foi quantificada, para um conjunto de sequências de vídeo de diferentes resoluções para um conjunto de quatro valores de parâmetro de quantização (Quantization Parameter – QP). O experimento foi realizado executando o software HM em uma plataforma com processador Intel Core i7-2600 com 16 GB de memória.

Neste resumo é mostrado o resultado do profiling para a sequência de vídeo de resolução ultra-alta PeopleOnStreet (com 2560x1600 pixels). A Figura A.1 apresenta a distribuição do tempo de execução para as ferramentas de codificação mais importantes do HEVC em termos de esforço computacional. No codificador HEVC, 55%-70% do tempo total de codificação corresponde a três ferramentas de codificação: Filtro de Interpolação de Pixel Fracionário, Soma das Diferenças Absolutas (SAD) e Soma das Diferenças Transformadas Absolutas (SATD). A contribuição no tempo depende do QP. No decodificador HEVC, 35%-55% do tempo de execução corresponde a duas ferramentas: Filtro de Interpolação de Pixel Fracionário e o Filtro de Deblocagem.

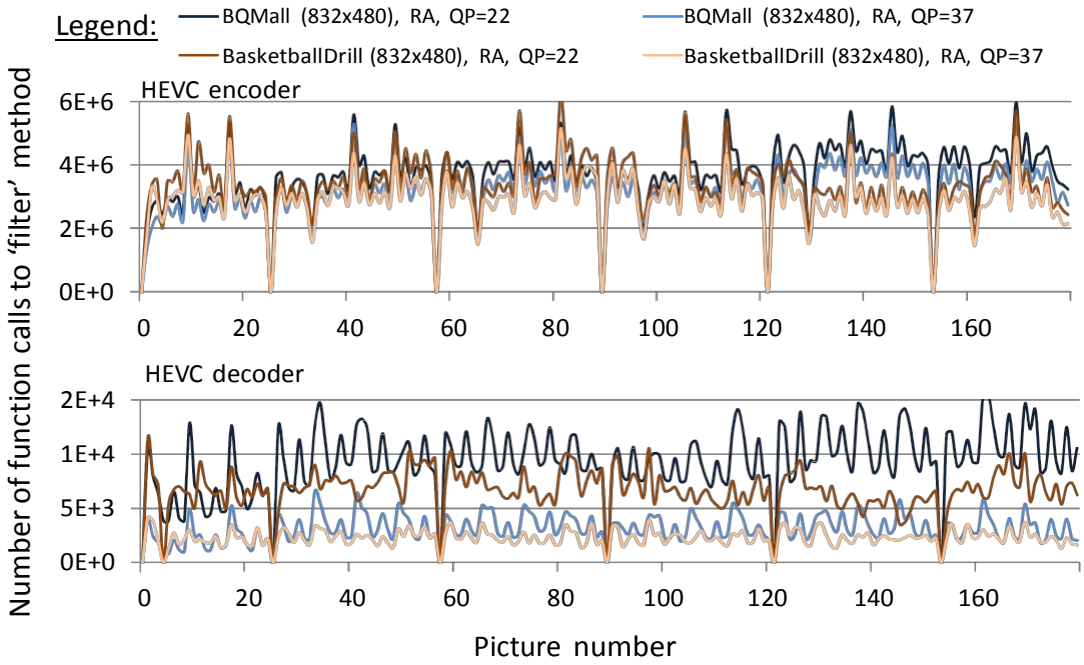
Figura A.1 – Contribuição de diferentes ferramentas de codificação do HEVC (percentual) do tempo de execução total. Sequência de vídeo: “People on Street” (2560x1600 pixels), 150 quadros



Fonte: o autor.

Com relação ao Filtro de Interpolação de Pixel Fracionário, foi realizada também uma análise de tempo de execução, como mostrado na Figura A.2. Esta análise mostra que o número de interpolações por imagem (quadro) varia significantemente mesmo para vídeos de mesma resolução e mesmo QP. Desta forma, existe um grande potencial de redução de potência em tempo real se for explorada esta variação de carga de trabalho (workload).

Figura A.2 – Número de chamadas por quadro à função básica do filtro de interpolação

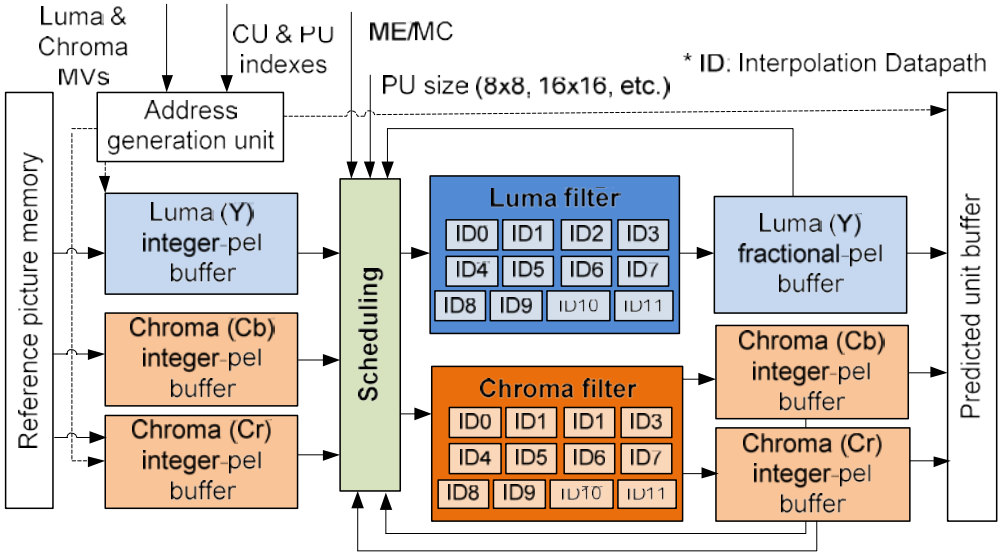


Fonte: o autor.

A.3. Aceleradores de Hardware Dedicados

Primeiramente, uma arquitetura dedicada para o Filtro de Interpolação de Pixel Fracionário foi proposta nesta tese. Esta arquitetura é mostrada na Figura A.3.

Figura A.3 – Diagrama da Arquitetura Proposta para o Filtro de Interpolação de Pixel Fracionário do HEVC



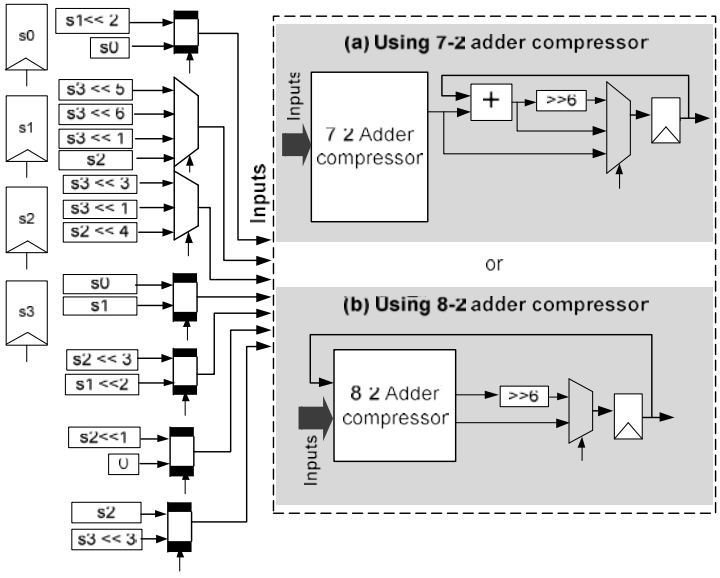
Fonte: (DINIZ, 2013).

A arquitetura é composta de dois núcleos de aceleração, um para luminância e outro para crominância. Cada núcleo é composto de 12 datapaths em paralelo para atingir a taxa de processamento desejada. Cada datapath é configurável para selecionar o tipo de filtragem, e substitui as multiplicações por constantes através de operações de soma e

deslocamento. Um módulo para escalonamento dos datapaths também é proposto nesta tese. Outros módulos apresentados na Figura A.3 não fazem parte do escopo do trabalho. Esta arquitetura reduz a área de hardware comparada com trabalho estado da arte. Mais detalhes desta arquitetura podem ser encontrados em (DINIZ, 2013).

Esta arquitetura também foi objeto do estudo de uma proposta para substituição dos somadores, presentes nos datapaths, por somadores compressores, que fornecem uma eficiência maior quando a soma de múltiplos operandos é necessária, o que é o caso desta arquitetura. A aplicação de somadores compressores 8-2 e 7-2 no datapath de luminância do filtro de interpolação de pixel fracionário do HEVC é mostrada na Figura A.4. Mais detalhes a arquitetura do filtro de interpolação usando somadores compressores pode ser consultada em (DINIZ, 2015c).

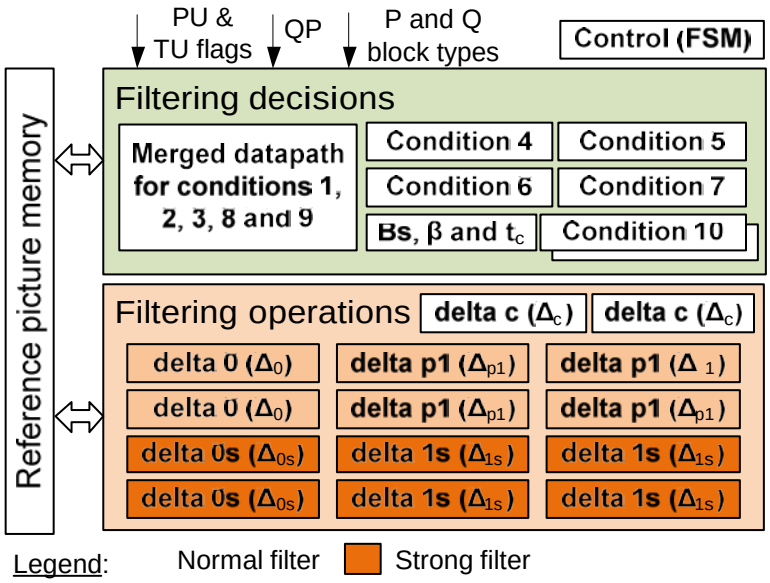
Figura A.4 – Arquitetura modificada do datapath do filtro de interpolação de luminância usando (a) somador compressor 7-2; (b) somador compressor 8-2.



Fonte: o autor.

Uma arquitetura para o Filtro de Deblocagem do HEVC é proposta nesta tese, sendo apresentada na Figura A.5. Ela contém dois módulos principais: o módulo de decisões de filtragem (pois trata-se de um filtro adaptativo) e o módulo de operações de filtragem (que inclui dois tipos de filtro, o normal e o forte). Cada módulo contém datapaths que calculam as decisões e operações conforme as equações de filtro definidas no padrão HEVC. Onde haviam multiplicações por constantes, estas foram substituídas por operações de soma e deslocamento. No módulo de decisões, foi possível realizar um só datapath para calcular as condições 1, 2, 3, 8 e 9, reusando hardware. Esta arquitetura reduz a área de hardware comparada com trabalhos estado da arte, mantendo a taxa de processamento. Mais detalhes sobre a arquitetura podem ser consultados em (DINIZ, 2015b).

Figura A.5 – Diagrama da arquitetura de hardware proposta para o filtro de deblocagem do HEVC



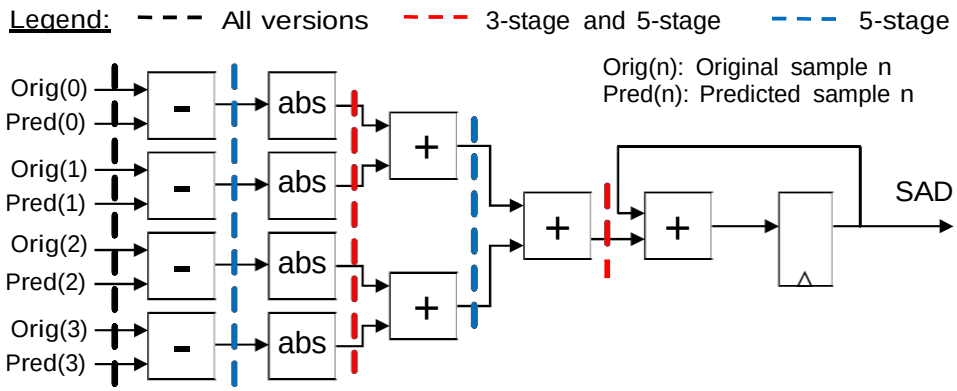
Fonte: (DINIZ, 2015b).

Esta seção também apresenta o desenvolvimento de alternativas arquiteturais para o elemento de processamento do cálculo da Soma das Diferenças Absolutas (SAD). Foram desenvolvidas nove alternativas arquiteturais, combinando dois parâmetros: i) o número de amostras de entrada em paralelo (4, 8 e 16); ii) o número de estágios de pipeline, dependendo de cada versão de paralelismo, conforme abaixo:

a) Alternativas com 1, 3 e 5 estágios de pipeline para o elemento de processamento de SAD com 4 amostras de entrada;
b) Alternativas com 1, 3 e 6 estágios de pipeline para o elemento de processamento de SAD com 8 amostras de entrada;
c) Alternativas com 1, 4 e 7 estágios de pipeline para o elemento de processamento de SAD com 16 amostras de entrada.

A Figura A.6 ilustra somente o elemento de processamento de SAD com 4 amostras de entrada. Mais detalhes sobre este trabalho podem ser consultados em (DINIZ, 2010).

Figura A.6 – Alternativas para arquitetura de SAD com 4 amostras de entrada



Fonte: (DINIZ, 2010).

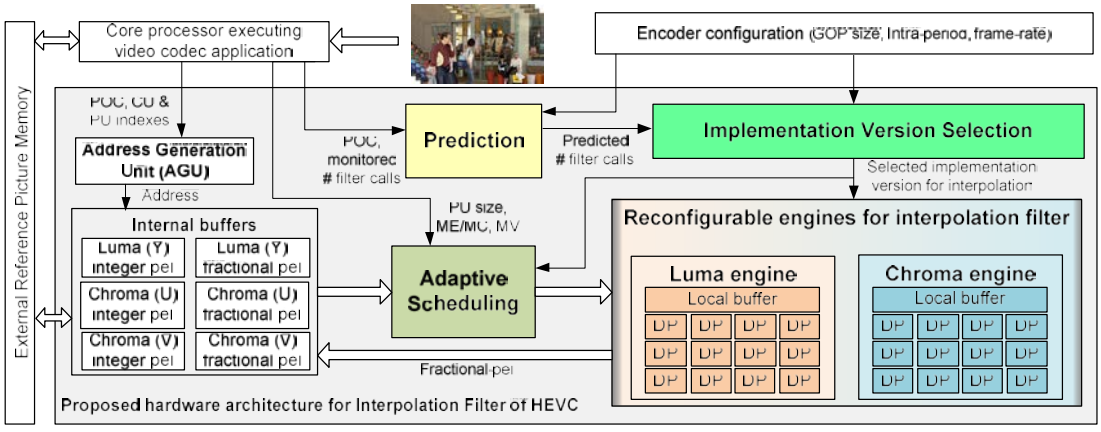
A.4. Arquitetura de Hardware Reconfigurável para o Filtro de Interpolação de Pixel Fracionário do HEVC

Esta tese propõe uma arquitetura reconfigurável para o Filtro de Interpolação de Pixel Fracionário, dado a análise da seção A.2 de que é possível economizar potência/energia em tempo de execução. A arquitetura é composta por quatro módulos principais:

1) Módulo de Predição que estima o número de chamadas do filtro de interpolação para próximos quadros a serem codificados, baseado no monitoramento de interpolações em GOPs passados.
2) Núcleos de Aceleração para Luma e Chroma. Filtros de interpolação em hardware com um conjunto de diferentes versões de implementação proporcionando um número de opções com diferentes resultados de área de desempenho.
3) Módulo de Seleção da Versão de Implementação que seleciona a versão de implementação adequada, reconfigurando a versão de implementação (número de datapaths em paralelo) baseado no número de chamadas de interpolação preditas pelo módulo de predição.
4) Módulo de Escalonamento Adaptativo que determina a ordem de processamento e a configuração do tipo de filtro de forma adaptativa.

O diagrama da arquitetura é mostrado na Figura A.7. Esta arquitetura assume que os núcleos de aceleração são reconfigurados em FPGA que suportam reconfiguração parcial dinâmica (XILINX, 2010) (ALTERA, 2010). Mais detalhes sobre esta arquitetura podem ser consultados em (DINIZ, 2015a).

Figura A.7 – Arquitetura Reconfigurável para o Filtro de Interpolação de Pixel Fracionário do HEVC

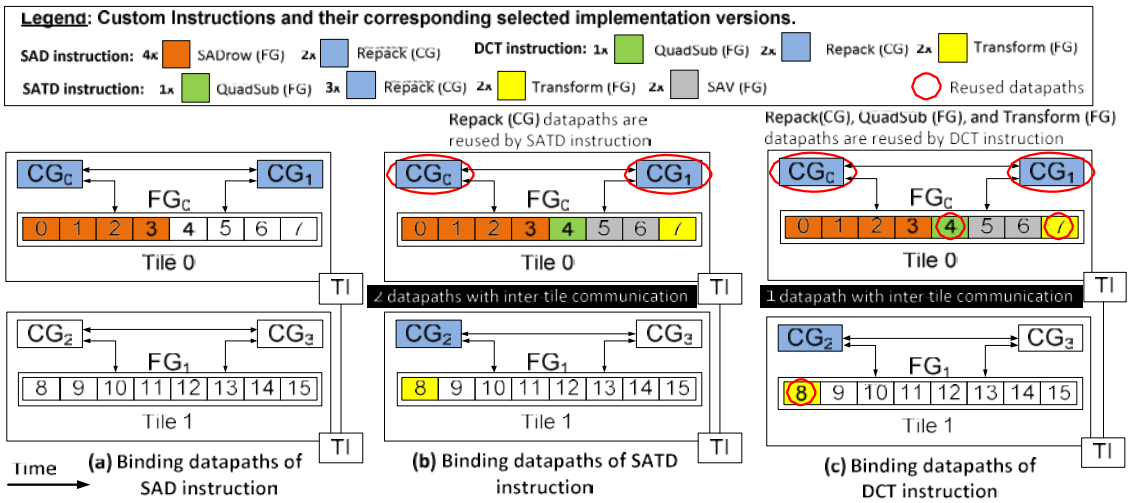


Fonte: (DINIZ, 2015a).

A.5. Alocação de Aceleradores em Tempo de Execução para Arquiteturas Reconfiguráveis

Esta seção apresenta o esquema de alocação em tempo de execução para arquiteturas reconfiguráveis de grão misto baseadas em tiles de processamento. Trata-se de um problema que não foi encontrada solução reportada na literatura. O problema do uso de uma estratégia trivial, não ciente da arquitetura, para alocação de aceleradores neste tipo de arquitetura é mostrado na Figura A.8.

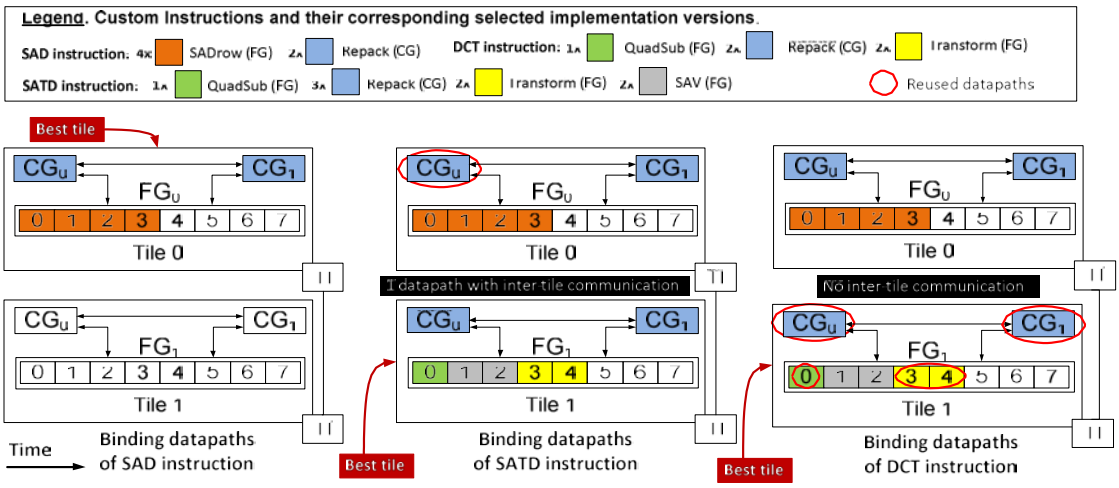
Figura A.8 – Exemplo de alocação de aceleradores para 3 instruções usando esquema first-fit com reuso de datapaths



Fonte: (DINIZ, 2014).

Este trabalho propõe um novo esquema para alocação de aceleradores (Figura A.9) que é ciente da arquitetura reconfigurável baseada em tiles. Ele aplica o reuso de datapaths, somente nos casos em que é benéfico devido à divisão de tiles. Ele também aplica estimação de comunicação quando é necessário alocar aceleradores em tiles diferentes. Mais detalhes podem ser consultados em (DINIZ, 2014).

Figura A.9 – Esquema de alocação proposto (usando mesmo exemplo anterior)



Fonte: (DINIZ, 2014).

A.6. Conclusões e Trabalhos Futuros

A presente tese focou na contribuição de aceleradores dedicados e reconfiguráveis inovadores para o padrão HEVC.

O trabalho de pesquisa começou com uma análise da aplicação de codificação HEVC, como apresentada na seção A.2, o que indicou que as ferramentas de codificação mais importantes a serem aceleradas em hardware são o Filtro de Interpolação de Pixel Fracionário, o Filtro de Deblocagem, e o cálculo da Soma das Diferenças Absolutas, necessária para Estimação de Movimento. Os resultados da análise mostraram que há variações significativas dependendo da sequência de vídeo a ser codificada (definida como entrada pelo usuário) e do parâmetro de quantização que define o nível de perda de dados na codificação. Uma análise de tempo real do Filtro de Interpolação indica que existe um grande potencial de economia de potência/energia através da adaptação do acelerador de hardware à carga de trabalho variável.

Os resultados obtidos dos aceleradores de hardware dedicados inovadores (seção A.3) indicam ganhos significativos sobre aceleradores de hardware do estado da arte. A arquitetura dedicada de hardaware para o Filtro de Interpolação atinge taxa de processamento suficiente para processar vídeos de resolução ultra-alta e reduz a área de hardware por cerca de 50% comparado com uma arquitetura estado da arte. Tal ganho foi obtido pelo projeto de datapaths configuráveis sem multiplicadores e eficientes em área. A taxa de processamento foi melhorada através do uso de dois núcleos de aceleração com nível de paralelismo de 12 pixels em paralelo, que contém os datapaths configuráveis. Um módulo de escalonamento foi projetado para prevenir bolhas no pipeline e para aprimorar a localidade de memória, reduzindo o uso de memória. A arquitetura proposta para o Filtro de Deblocagem atinge taxa de processamento similar com arquiteturas do estado da arte, enquanto reduz o número de portas (gate count) em 5 a 6 vezes, e reduz a potência em 3 vezes comparado com estas arquiteturas. Os datapaths desenvolvidos neste trabalho são altamente otimizados para área em utilizam reuso de hardware. Nossa análise comparativa de elementos de processamento para o SAD introduziu várias alternativas arquiteturais para explorar diferentes compromissos de área, desempenho e potência.

A arquitetura reconfigurável para o filtro de interpolação de pixel fracionário do padrão HEVC, descrita na seção A.4, é nova e proporciona significativa redução de área em tempo de projeto e adaptação de potência/energia em tempo de execução a cada quadro do vídeo. Esta característica não era ainda suportada pelas arquiteturas do filtro de interpolação do estado da arte. A adaptação em tempo de execução é realizada através de um esquema de predição, que estima o número de chamadas ao filtro de interpolação e um módulo de seleção de versão de implementação que adapta a diferentes taxas de processamento pela seleção de diferentes versões de implementação.

O esquema de alocação de aceleradores em tempo de execução para arquiteturas reconfiguráveis de grão misto baseadas em tiles de processamento, apresentado na seção A.5, reduz o overhead de comunicação, comparado com uma estratégia first-fit com reuso de datapaths, em até 44% (23% em média) para diferentes números de tiles e diferentes organizações internas de tiles. Este esquema de alocação é ciente da arquitetura baseada em tiles, para alocar de forma eficiente os aceleradores, evitando comunicações de aceleradores entre dois ou mais tiles diferentes.

Os resultados no geral demonstraram que os novos aceleradores dedicados e reconfiguráveis propostos nesta tese estão à frente de soluções do estado da arte. Devido

às limitações de potência e energia das tecnologias CMOS atuais e aos altos requisitos de desempenho dos padrões de codificação de vídeo da nova geração, futuras implementações de sistemas de codificação de vídeo irão integrar, no mesmo chip, processadores de muitos núcleos (many-core) com muitos aceleradores dedicados e reconfiguráveis, nas chamadas arquiteturas ricas em aceleradores.

As arquiteturas ricas em aceleradores são necessárias para codificação de vídeo de ultra-alta resolução em tempo real com eficiência em potência/energia. Neste contexto, esta tese introduz novos aceleradores e técnicas que possibilitam implementações de codificação de vídeo de nova geração com aperfeiçoamento em área, desempenho, e eficiência em potência/energia.

Além das contribuições apresentadas nesta tese, várias direções de pesquisa emergem para o futuro, as quais não foram tratadas neste trabalho. Algumas destas direções de pesquisa são sugeridas abaixo como trabalhos futuros.

Aceleradores de hardware para outras ferramentas de codificação do HEVC: enquanto esta tese focou nas ferramentas de codificação mais intensivas em computação do HEVC, há outras ferramentas de codificação não tratadas que podem ser implementadas como aceleradores de hardware para um sistema completo de codificação de vídeo, tais como o cáculo da Soma das Diferenças Transformadas Absolutas (SATD), Codificação Binária Aritmética Adaptativa ao Contexto (CABAC), Predição Intra, Transformadas, Quantização, e o filtro de Offset de Amostra Adaptativo (Sample Adaptive Offset - SAO). O desafio de pesquisa é projetar aceleradores eficientes em desempenho/área/potência para estas ferramentas de codificação, comparadas a outras soluções presentes na literatura.

Exploração das ferramentas de codificação paralelas do HEVC: o HEVC inclui algumas ferramentas de codificação para facilitar o processamento paralelo para codificação de vídeo, tais como os Tiles e o Wavefront Parallel Processing (WPP). Com relação aos tiles, há muitos desafios de pesquisa que afetam tanto o desempenho como a qualidade do vídeo. O número de tiles para cada imagem e onde as fronteiras dos tiles são colocadas é decidido no lado do codificador do vídeo e não são padronizadas. Quebrar imagens em mais tiles aumenta a escalabilidade para processadores de muitos núcleos, mas degrada a qualidade do vídeo. Portanto, existe um compromisso entre o desempenho e qualidade do vídeo quando se usa tiles. Com relação ao WPP (quando algumas CTUs de uma imagem podem ser processadas em paralelo em uma abordagem multi-thread), o desafio de pesquisa é decidir em quais situação é benéfico utilizar dela. Existe também a decisão entre usar tiles ou WPP para cada imagem, porque o padrão HEVC ainda não suporta a coexistência destas duas ferramentas.

Aceleradores de hardware para processamento de imagens e vídeo: algumas metodologias são usadas neste trabalho para projetar aceleradores de hardware eficientes para ferramentas específicas do padrão HEVC. Estas metodologias podem ser aplicadas para projetar aceleradores para outras aplicações de imagens e vídeo, como processamento de imagens (filtragem e interpolação, por exemplo), codificação de imagens, pré-processamento de vídeo, etc.

Arquiteturas ricas em aceleradores para a era do Dark Silicon: os benefícios dos aceleradores de hardware comparados a processadores de propósito geral estão levando a pesquisa de processadores multi-core e many-core para a direção de arquiteturas ricas em aceleradores, ou seja, acoplar muitos núcleos de processamento com muitos

aceleradores de hardware dedicados e reconfiguráveis para funções específicas das aplicações. Neste tópico de pesquisa, muitos desafios são encontrados. Primeiramente, desafios de pesquisa em tempo de projeto devem ser abordados. Quais aceleradores a ser projetados e se eles devem ser projetados como aceleradores dedicados ou reconfiguráveis são boas questões de pesquisa. Perfilhamento da aplicação pode ajudar os projetistas a escolher quais aceleradores a projetar. Funções importantes usadas em muitas aplicações e/ou em sucessivas gerações de aplicações, são boas candidatas para os aceleradores dedicados. Aceleradores reconfiguráveis podem ser usados para proporcionar flexibilidade a funções importantes e emergentes (como o novo filtro de interpolação de pixel fracionário do HEVC, por exemplo) e para mapear outros aceleradores que são usados em algumas fases da execução da aplicação (devido ao fato de que aceleradores reconfiguráveis dissipam mais potência). No nível mais alto do projeto de sistema, outro desafio é como acoplar muitos aceleradores de hardware em muitos núcleos de processamento, e como conectá-los. Decisões em tempo de projeto são limitadas pela área do chip e potência máxima. Outro desafio de pesquisa emerge no contexto do controle em tempo de execução: como lidar com o desbalanceamento de carga de trabalho nas threads que executam em uma arquitetura rica em aceleradores. Um sistema de tempo de execução é essencial para alocar threads e ligar/desligar os aceleradores, com o principal objetivo de sustentar o desempenho dado um limite máximo de potência. O sistema de tempo de execução deve ser alimentado pelo monitoramento da aplicação e predição para otimizar o desempenho final da aplicação em potência e tempo.