Efficient hardware designs of the new Versatile Video Coding (VVC) tools for ASIC platforms

Brahim Farhat

THESE DE DOCTORAT DE

L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES RENNES

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Signal, Image et Vision*

Par

« Ibrahim FARHAT »

« Efficient hardware designs of the new Versatile Video Coding (VVC) tools for ASIC platforms »

Thèse présentée et soutenue à « Rennes », le « 31 Mai 2022 »
Unité de recherche : Institut d'Electronique et des Technologies du numéRique (IETR)- UMR CNRS 6164
Thèse N° : D22 - 11 / 22ISAR 11

Rapporteurs avant soutenance :

Fernando PESCADOR	Professeur	Université Polytechnique de Madrid (UPM)
Virginie FRESSE	Maitre de conférences	Université Jean Monet Saint-Etienne

Composition du Jury :
Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	Dominique GINHAC	Professeur des universités, Université de Bourgogne
Examinateurs :	Dominique GINHAC	Professeur des universités, Université de Bourgogne
	François BERRY	Professeur des universités, Université Clermont Auvergne
	Wassim HAMIDOUCHE	Maître de conférences, INSA Rennes
	Adrien GRILL	Ingénieur R&D, VITEC
Dir. de thèse :	Daniel MENARD	Professeur des universités, INSA Rennes
Co-dir. de thèse :	Olivier DEFORGES	Professeur des universités, INSA Rennes

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

INSA RENNES

Intitulé de la thèse :

Conceptions matérielles efficaces pour des nouveaux outils du standard VVC pour les plates-formes ASIC

–

Efficient hardware designs of the new Versatile Video Coding (VVC) tools for ASIC platforms

Acknowledgements

First of all, I would like to thank my family and friends for their support at all stages of my education to whom I owe every success in my life.

I would like to express my sincere and deep gratitude to my thesis supervisor at the IETR laboratory of INSA Rennes, Dr. Wassim Hamidouche, for his continuous support, his advice, his encouragement and for being always available when I needed it. I would like to thank him for his time, his support and his ideas.

I would also like to thank my thesis director at the IETR laboratory of INSA Rennes, Professor Daniel Menard for his advice, dedication and support during these years of work. I will always be grateful to him for giving me the opportunity to work with him.

I would like to thank my supervisor, Adrien Grill, from VITEC. He was able to convey the enthusiasm of his daily work by feeding it with daily challenges. In addition to being an excellent supervisor, he is an extremely diligent engineer. I would like to thank him for his time, his support and his ideas.

A special mention to Professor Olivier Déforge, my supervisor at IETR, for his support, time, wise advises and ideas. A special mention to Jean-Marc Thesis, my supervisor at VITEC, for his contribution of time, support and ideas.

'Unless you try to do something beyond what you have already mastered you will never grow.'
Ralph Waldo Emerson

Table of contents

List of figures

List of tables

Glossary

ALF Adaptive Loop Filter.

ALM Adaptive Logic Module.

APS Adaptation Parameter Set.

ASIC Application-Specific Integrated Circuit.

AVC Advanced Video Coding.

B bottom.

BD-BR Bjøntegaard Delta Rate.

BL bottom left.

BR bottom right.

BT Binary Tree.

CCALF Cross-Component Adaptive Loop Filter.

CM Constant Multipliers.

CTB Coding Tree Block.

CTC Common Test Conditions.

CTU Coding Tree Unit.

CU Coding Unit.

DC Design Compiler.

DCT Discrete Cosine Transform.

DF Deblocking Filter.

DPB Decoded Picture Buffer.

DSF Diamond Shape Filter.

DSP Digital Signal Processing.

DST Discrete Sine Transform.

FPGA Field Programmable Gate Array.

FPS Frame Per Second.

HEVC High Efficiency Video Coding.

HW Hardware.

IICT Inverse Integer Core Transforms.

IP Intellectual Property.

JEM Joint Exploration Model.

JVET Joint Video Experts Team.

KLT Karhunen-Loéve Theorem.

L left.

LFNST Low Frequency Non-Separable Transform.

MC Motion Compensation.

MPEG Motion Picture Experts Group.

MSE Mean Squared Error.

MTS Multiple Transform Selection.

MTT Multi Type Tree.

NSST Non-Separable Secondary Transform.

QP Quantization Parameter.

QT Quad-Tree.

R right.

RAM Random-Access Memory.

RD Rate Distortion.

RDO Rate Distortion Optimization.

RM Regular Multiplier.

ROM Read-Only Memory.

SAO Sample Adaptive Offset.

SPS Sequence Parameter Set.

SRAM Static Random Access Memory.

SSD Sum of Square Differences.

T top.

TL top left.

TR top right.

TSMC Taiwan Semiconductor Manufacturing Company.

TT Ternary Tree.

TU Transform Unit.

VB Virtual Boundary.

VCEG Video Coding Experts Group.

VHDL VHSIC Hardware Description Language.

VLSI Very Large Scale Integration.

VTM VVC Test Model.

VVC Versatile Video Coding.

Chapter 1

Résumé en Français

1.1 Préamble

Les récents progrès technologiques des appareils électroniques tels que les appareils mobiles, les écrans et les caméras ont fait exploser la demande de consommation multimédia. L'évolution rapide des processeurs puissants et des réseaux à haut débit, accompagnée d'un changement radical vers une interaction sociale en ligne croissante, conduit à une consommation vidéo d'une ampleur sans précédent. Selon une étude récente menée par la société Cisco, le trafic vidéo sur l'IP mondial représentera 82 % de l'ensemble du trafic IP d'ici 2023, contre 61 % en 2016 [7]. Une autre étude de la société Ericsson annonce que le trafic vidéo devrait représenter 69 % de l'ensemble du trafic de données mobiles en 2021 et devrait passer à 79 % en 2027 [8]. En outre, les applications vidéo émergentes telles que les vidéos à gamme dynamique élevée (HDR) et les vidéos à 360° nécessitent une consommation de bande passante élevée et des recherches supplémentaires.

La norme HEVC [9] a été finalisée en 2013 pour répondre à la demande croissante de vidéos à plus haute résolution, apportant un gain de compression de 50% à qualité égale par rapport à son prédécesseur H.264/Advanced Video Coding (AVC). Contestée par le codec ouvert et libre de droits AV1, une nouvelle norme de codage vidéo VVC [10] a été finalisée en juillet 2020, apportant un gain de compression supplémentaire par rapport au HEVC. VVC comprend des outils de codage supplémentaires pour répondre à la consommation vidéo jusqu'à 8K ou plus, au HDR, à la gamme de couleurs étendue, au contenu d'écran généré par ordinateur et à la vidéo 360° pour la réalité virtuelle et augmentée. Toutes les normes de compression vidéo traitent la transmission efficace et le stockage numérique des vidéos. La procédure de normalisation englobe diverses exigences, notamment le débit binaire, la qualité de l'image, le délai, l'accessibilité aléatoire et la complexité. Une norme de codage vidéo est conçue pour répondre à de nombreuses applications, á savoire les communications en temps réel, la diffusion de vidéos à la demande et la diffusion en direct. Cependant, ces normes ne spécifient pas directement une mise en œuvre pour un produit final. L'application d'une norme à un produit nécessite une adaptation soigneuse à l'application cible spécifique. En particulier, l'application en temps réel présente de nombreuses contraintes auxquelles il faut répondre avant qu'elle ne devienne un produit valide. L'utilisation d'une solution Hardware (HW) dédiée permet d'atteindre une qualité supérieure tout en maintenant une faible consommation d'énergie et une transmission en temps réel.

La conception et l'implémentation d'une solution HW pour le nouveau standard VVC est le terrain de jeu de cette thèse.

1.2 Contexte et motivation

Cette thèse se concentre sur la conception et l'implémentation de nouveaux outils VVC pour une application en temps réel ciblant un encodeur et un décodeur ASIC HW professionnel disponible comme produit commercial sur le marché. L'objectif de ces produits est de supporter des applications d'encodage et de décodage en temps réel jusqu'à 4K tout en maintenant une faible consommation d'énergie dans un dispositif matériel compact dédié, entièrement développé par VITEC, la société partenaire française experte en codage vidéo en temps réel.

VVC introduit de nouveaux outils de codage qui améliorent l'efficacité globale du codage par rapport à son prédécesseur HEVC. Cependant, ces nouveaux outils ont un impact significatif sur la complexité du VVC qui est estimée à quatre fois celle du HEVC. Ces outils s'accompagnent d'une forte augmentation de la complexité de calcul du côté du décodage et (encore plus) du codage. En particulier, la partie transformée et le filtre en boucle de VVC introduisent leurs propres nouveaux outils et ont un impact majeur sur les complexités d'encodage et de décodage. Le module de transformation VVC comprend des transformations séparables et non séparables nommées respectivement MTS et LFNST. L'outil MTS implique trois types de transformations trigonométriques, notamment DCT type 2 (DCT-2), DCT type 8 (DCT-8) et DST type 7 (DST-7), avec des tailles de bloc pouvant atteindre 64×64 pour DCT-2 et 32×32 pour DCT-8 et DST-7. L'utilisation des familles DCT/DST donne la possibilité d'appliquer des transformations séparables, la transformation d'un bloc pouvant être appliquée séparément dans les directions horizontale et verticale. La LFNST, quant à elle, est une transformation non séparable, ce qui signifie qu'elle est appliquée en une seule fois. Et ceci après la transformation séparable et avant la quantification du côté de l'encodeur. Du côté du décodeur, elle est appliquée après la quantification inverse et avant la transformation séparable inverse. Différente de la transformation primaire, la transformation secondaire n'opère que sur les coefficients de transformation basse fréquence. Elle est appliquée pour toutes les tailles de Transform Unit (TU) de 4×4 à 64×64, y compris les blocs symétriques et asymétriques. Pour son processus de filtre en boucle, VVC introduit l'utilisation d'un nouvel outil appelé ALF. Les filtres en boucle sont situés dans la boucle de décodage de l'encodeur et visent à améliorer la qualité perçue de la séquence vidéo décodée en éliminant les artefacts de blocage, d'anneau ou de flou générés par les étapes de décodage précédentes. Le rôle principal du ALF est de contribuer à la réduction des artefacts visibles tels que le ringing et le blurring en diminuant l'erreur absolue moyenne entre l'image originale et l'image reconstruite. Pour ce faire, il utilise différents filtres pour les composantes Luma et Chroma. Pour la composante Luma, il utilise un 7×7 Diamond Shape Filter (DSF). Pour les composantes Chroma, le ALF utilise deux filtres. Le premier est un filtre DSF 5×5 et le second est le filtre diamant CCALF qui utilise les échantillons Luma co-localisés pour filtrer les échantillons Chroma. De plus, l'utilisation d'une conception HW dédiée au codage vidéo s'avère plus efficace que la solution logicielle, selon l'étude publiée par Xilinx [11]. En outre, un HW codec efficace doit être optimisé en termes de taille, de poids et de consommation d'énergie afin d'offrir la meilleure expérience utilisateur tout en réalisant

une application en temps réel avec la meilleure qualité vidéo.

1.3 Contributions

L'objectif global de ce doctorat est d'étudier les conceptions matérielles et la réduction de la complexité des nouveaux outils VVC dans un cadre de codage en temps réel pour les plateformes ASIC. Les conceptions proposées visent à réduire le coût de la puce ASIC en termes de surface, de mémoire et de consommation électrique. Trois outils principaux ont été étudiés pour le contexte de l'encodeur et du décodeur : MTS, LFNST et ALF. Dans ce contexte, les contributions de la thèse ont été identifiées comme suit :

- **Conception matérielle Transformer pour le VVC encodeur et décodeur**: L'objectif est de concevoir une implémentation matérielle efficace pour les outils de transformation dans le contexte d'un décodeur et d'un encodeur ciblant la plate-forme ASIC. Il s'agit d'une architecture multistandard qui prend en charge le bloc de transformation des normes MPEG récentes, notamment AVC, HEVC et VVC. L'architecture est optimisée et élimine les complexités inutiles trouvées dans d'autres architectures proposées en utilisant des multiplicateurs réguliers au lieu de multiplicateurs constants. Le LFNST, un des nouveaux outils introduits dans le VVC, est implémenté dans une conception unifor avec le MTS. La conception proposée utilise un total de 64 multiplicateurs réguliers dans une architecture pipe-lined. Il exploite toutes les optimisations des transformations primaires et secondaires, y compris la décomposition papillon, la remise à zéro des coefficients et la relation linéaire inhérente entre les types de transformation. La transformation unifiée maintient un débit constant de 1 échantillon par cycle et une latence système fixe pour toutes les tailles de blocs.

- **Conception matérielle ALF pour VVC encodeur et décodeur**: L'objectif est de concevoir une implémentation matérielle efficace pour le nouvel outil ALF dans le contexte d'un décodeur et d'un encodeur ciblant une plateforme ASIC 4K. Une nouvelle architecture matérielle est présentée, en tenant compte de ses trois filtres : 7×7, 5×5 et 3×4 CCALF. La solution proposée établit un ordre de balayage entre les composantes Luma et Chroma qui réduit considérablement la mémoire. La conception tire parti de toutes les caractéristiques de ALF et établit un module matériel unifié pour tous les filtres. Elle utilise 26 multiplicateurs réguliers dans une architecture pipe-lined avec un débit fixe de 2 pixels/cycle et une latence système fixe quel que soit le filtre sélectionné. Pour le contexte du codeur, le ALF comporte deux phases : le filtrage et le processus de décision. Il a été conçu après une étude approfondie de l'algorithme de décision ALF et a abouti à une architecture matérielle de première génération, mais implémentable, capable de capter une part importante des performances de cet outil. En fait, l'étude a révélé qu'un bon compromis entre le taux, la distorsion et la complexité matérielle est trouvé lorsque le codeur active le ALF pour les échantillons Luma uniquement et pour les CTUs de taille 64. En outre, il limite le processus de décision à quatre ensembles fixes de coefficients, testant ainsi quatre des seize indices d'ensembles de filtres définis dans la norme. À notre connaissance,

il s'agit de la première contribution dans la littérature qui propose une étude de conception matérielle complète pour le module ALF.

Dans cette thèse, les conceptions et études proposées pour ces objectifs ont été intégrées dans un encodeur/décodeur vidéo professionnel en temps réel HW.

1.4 Organisation

Le chapitre 3 commence par une introduction aux bases du codage vidéo et à certains concepts essentiels pour cette thèse, de l'histoire de la compression vidéo à travers les principaux standards et codecs, au schéma classique de compression vidéo hybride. Il présente un bref aperçu des principaux outils de l'état de l'art VVC. En outre, il présente la caractérisation d'un codec vidéo en temps réel avec une comparaison entre plusieurs codeurs commerciaux.

Dans le chapitre 4, l'accent est rapidement mis sur l'étape de transformation où un historique du processus de transformation est discuté pour les normes VVC. En particulier, les outils MTS et LFNST sont discutés et décrits en détail. Ensuite, les propositions importantes de l'état de l'art en matière d'optimisation de transformée, d'approximation et d'implémentation matérielle essentiellement pour les types de transformée MTS sont présentées.

Le chapitre 5 aborde l'implémentation du processus MTS y compris ses trois types de transformation (DCT-2, 8 et DST-7). Ce dernier est conçu dans le contexte d'un système 4K HW temps réel. ASIC VVC en temps réel. Dans ce cas, le module 2-D MTS doit soutenir un débit fixe de 1 échantillon/cycle. En outre, pour faciliter le chaînage entre différentes tailles de blocs, ce dernier doit maintenir une latence unifiée indépendamment des tailles de blocs et des types de transformation. Presque toutes les études de l'état de l'art pour la conception de la première transformation HW considèrent principalement une approche sans multiplicateur, c'est-à-dire CM, au lieu d'utiliser RM. Dans ce travail, une étude et une comparaison pour deux architectures matérielles pour le VVC MTS est présentée. La première implémente la transformation en utilisant une architecture basée sur CM, tandis que la seconde utilise une architecture basée sur RM. Enfin, les résultats expérimentaux et de synthèse sont discutés et comparés aux travaux connexes.

Le chapitre 6 aborde le problème de l'implémentation matérielle du VVC 1D inverse. MTS et LFNST visant une résolution de 4Kp60 et une fréquence de 450Mhz. Les MTS et LFNST 1-D doivent tous deux supporter une latence système fixe avec un débit fixe de 4 échantillons/cycle. D'après notre étude précédente du chapitre 5, la meilleure approche pour implémenter l'iMTS est une architecture basée sur l'RM. C'est également la même approche utilisée pour mettre en œuvre l'iLFNST. Dans ce travail, une architecture matérielle unifiée pour les VVC iMTS et iLFNST utilisant une architecture basée sur RM est décrite. Enfin, le montage expérimental et les résultats de synthèse des implémentations de la transformation VVC avec une comparaison avec la transformation HEVC sont présentés.

Le chapitre 7 présente une comparaison entre une conception partagée unissant le DCT-2 et son inverse et une conception partagée unissant le DCT-2 et le LFNST à un débit de 32 échantillons/cycle dans le contexte d'un VVC 4K temps réel. ASIC en temps réel. Tout d'abord, le coût de chaque

conception partagée séparément est étudié. Ensuite, une comparaison entre les deux solutions avec une mise en évidence des principales différences est présentée.

Le chapitre 8 se concentre sur le contexte de l'outil ALF de VVC. Il présente les différents filtres utilisés par l'ALF pour les composantes Luma et Chroma. En outre, il présente les implémentations matérielles existantes de l'ALF.

Le chapitre 9 présente une nouvelle architecture matérielle pour le filtre VVC ALF, en considérant ses trois filtres : 7×7, 5×5 et 3×4 CCALF. Cette conception est basée sur le dernier Working Draft et le dernier VVC Test Model (VTM). La conception proposée maintient un débit fixe de 2 pixels/cycle avec une latence système fixe qui permet le traitement de vidéo 4K à 60 images par seconde en sous-échantillonnage 4:2:2 Chroma. La conception proposée est intégrée dans une puce décodeur vidéo professionnelle supportant plusieurs normes de codage vidéo. L'article décrit tout d'abord la conception de haut niveau du module ALF. Deuxièmement, il montre les différents balayages d'entrée ainsi que l'ordre de balayage le plus optimal. Troisièmement, il décrit les configurations de mémoire de l'ordre de balayage d'entrée proposé. Ensuite, la conception proposée pour le module de gradient et pour les filtres ALF de base est abordée. Enfin, il présente le montage expérimental et les résultats de synthèse du ALF ainsi qu'une comparaison avec l'état de l'art.

Le chapitre 10 se concentre sur la conception matérielle des ALF dans le contexte du codeur. Il présente tout d'abord les conceptions matérielles des modules de mémoire, de somme des gradients et de filtrage pour le ALF Luma pour un débit de deux pixels/cycle. Deuxièmement, il décrit l'architecture de décision ALF pour un débit d'un pixel/cycle. Troisièmement, il présente les conceptions matérielles de la mémoire, du gradient, du filtrage et des modules SSD pour la décision ALF. Enfin, il présente le montage expérimental et les résultats de synthèse en comparaison avec le décodeur ALF.

Chapter 2

Introduction

2.1 Preamble

The recent technological progress in electronic devices such as mobile devices, screens and cameras has skyrocketed the multimedia consumption demand. Rapidly evolving powerful processors and high-speed networks accompanied with a radical shift to increasing online social interaction lead video consumption on an unprecedented scale. According to a recent study conducted by Cisco company in [7], the video traffic over Global IP will account for 82% of all IP traffic by 2023, up from 61% in 2016. Another study from Ericsson company announced that video traffic is estimated to account for 69% of all mobile data traffic in 2021 and is expected to increase to 79% percent in 2027 [8]. Besides, emerging video applications such as High Dynamic Range (HDR) and 360 videos require a high bandwidth consumption and further research.

The HEVC standard [9] was finalized in 2013 to address growing demands for higher resolution videos, bringing a 50% compression gain at the same quality against its predecessor H.264/AVC. Challenged by the open and royalty-free codec AV1, a new video coding standard VVC [10] was finalized in July 2020, bringing further compression gain over HEVC. VVC includes additional coding tools to address video consumption up to 8K or larger, HDR, wide color gamut, computer-generated screen content, and 360° video for virtual and augmented reality. All video compression standards address efficient transmission and digital storage of videos. The standardization procedure encapsulates various requirements, including bit rate, picture quality, delay, random accessibility, and complexity. A video coding standard is designed to address many applications, including real-time communications, on-demand video delivery, and live streaming. However, these standards do not directly specify an implementation for an end product. Applying a standard to a product requires careful adaptation to the specific target application. In particular, the real-time application has many constraints that must be answered before it can become a valid product. Using dedicated HW solution allows the capability to reach higher quality while maintaining low energy consumption and real-time transmission. Designing and implementing HW solution for the new VVC standard is the playground of this thesis.

2.2 Context and Motivation

This thesis focuses on designing and implementing new VVC tools for a real-time application targeting a professional HW ASIC encoder and decoder available as a commercial product in the market. The aim of these products is to support up to 4K real-time encoding and decoding applications while maintaining a low power consumption in a compact dedicated hardware device fully developed by VITEC, the French partner company which expertise in real-time video coding.

VVC introduces new coding tools that enhances the overall coding efficiency compared to its predecessor HEVC. It is designed to deliver a 30% to 50% bit-rate reduction as compared to HEVC at the same perceptual quality with a heavy computational complexity increase on both decoding and (even more) encoding side. It is estimated that VVC has an encoding complexity of 10× HEVC and a decoding complexity of around 2×. In particular, the transform and the in-loop filter parts of VVC introduces their own new tools and impacts in a major way the encoding and decoding complexities. The VVC transform module includes separable and non-separable transforms named MTS and LFNST, respectively. The MTS tool involves three trigonometrical transform types including DCT type 2 (DCT-2), DCT type 8 (DCT-8) and DST type 7 (DST-7) with block sizes that can reach 64×64 for DCT-2 and 32×32 for DCT-8 and DST-7. The use of DCT/DST families gives the ability to apply separable transforms, the transformation of a block can be applied separately in horizontal and vertical directions. The LFNST on the other hand is a non-separable transform, which means that it is applied in a single pass. It is applied after the separable transform and before the quantisation at the encoder side. At the decoder side, it is applied after the inverse quantisation and before the inverse separable transform. Different from the primary transform, the secondary transform operates only on the low-frequency transform coefficients. It is applied for all TU sizes from 4×4 to 64×64 including symmetric and asymmetric blocks. For its in-loop filter process, VVC introduces the use of a new tool called ALF. In-loop filters are located in the decoding loop of the encoder and aim to enhance the perceived quality of the decoded video sequence by eliminating the blocking, ringing or blurring artifacts generated by previous decoding stages. The main role of the ALF is to contribute in reducing visible artifacts such as ringing and blurring by decreasing the mean absolute error between the original image and the reconstructed one. To do so, it uses different filters for Luma and Chroma components. For the Luma, it uses a 7×7 DSF. For the Chroma components, the ALF uses two filters. The first one is a 5×5 DSF filter and the second one is the CCALF diamond filter which uses the co-located Luma samples to filter the Chroma ones. On the other hand, the use of dedicated HW design for video coding is proven to be more efficient compared to software-based solution according to the study published by Xilinx [11]. In addition, a successful HW codec needs to be optimized in term of size, weigh and power consumption in order to reach the best user experience while achieving real-time application with the highest video quality.

2.3 Manuscript contributions

The overall goal of this Ph.D. is to study the hardware designs and complexity reduction of new VVC tools in a real-time coding framework for ASIC platforms. The proposed designs aims to reduce

the cost of the ASIC chip in term of area, memory and power consumption. Three main tools were studied for the context of encoder and decoder: MTS, LFNST and ALF. With the above context, the contribution of the thesis have been identified as follow:

- **Transform hardware design for VVC ASIC chip**: the goal is to design an efficient hardware implementation for the transform tools in the context of decoder and encoder targeting ASIC platform. It consists in a multi-standard architecture that supports the transform block of recent MPEG standards including AVC, HEVC and VVC. The architecture is optimized and removes unnecessary complexities found in other proposed architectures by using regular multipliers instead of constant multipliers. The LFNST, one of the new tools introduced in VVC, is implemented in a uniform design along with the MTS. The proposed design uses a total of 64 regular multipliers in a pipe-lined architecture. It leverages all primary and secondary transforms optimisations including butterfly decomposition, coefficients zeroing and the inherent linear relationship between the transform types. The unified transform sustains a constant throughput of 1 sample per cycle and a fixed system latency for all block sizes.

- **ALF Hardware Design for VVC ASIC chip**: the goal is to design an efficient hardware implementation for the new ALF tool in the context of decoder and encoder targeting a 4K ASIC platform. A novel hardware-architecture is presented, considering its three filters: 7×7, 5×5 and 3×4 CCALF. The proposed solution establishes a scanning order between Luma and Chroma components that decreases significantly the memory. The design takes advantage of all ALF features and establishes an unified hardware module for all the filters. It uses 26 regular multipliers in a pipe-lined architecture with a fixed throughput of 2 pixels/cycle and a fixed system latency regardless of the selected filter. For the encoder context, the ALF has two phases: the filtering and the decision process. It has been designed after an in-depth study of the ALF decision algorithm and ended with a first generation hardware architecture, yet implementable, able to catch a significant amount of this tool performance. In fact, the study revealed that a good compromise between rate, distortion and hardware complexity is found when the encoder activates the ALF for Luma samples only and for CTUs of size 64. In addition, it limits the decision process on four fixed sets of coefficients, thus, testing four out of sixteen filter set indices defined in the standard. To the best of our knowledge, this is the first contribution in the literature that proposes a full hardware design study for the ALF module.

In this thesis, the proposed designs and studies for these objectives have been integrated into a real-time professional HW video encoder/decoder.

2.4 Manuscript organization

This section presents the overall organization of this thesis.

Chapter 3 starts with an introduction to the basics of video coding and some essential concepts for this thesis, from the history of video compression across the main standards and codecs, to the classical hybrid video compression scheme. It presents a brief overview of the main tools of the state-

of-the art VVC. In addition it shows the characterization of a real time video codec with a comparison between several commercial encoders.

In Chapter 4, the focus is quickly put on the transform stage where a background of transform process is discussed for the VVC standards. In particular, the MTS and LFNST tools are discussed and described in details. Then, important existing state-of-the-art proposals in transform optimization, approximation and hardware implementation essentially for MTS transform types are presented.

Chapter 5 addresses the implementation of the MTS process including its three transform types (DCT-2, 8 and DST-7). This latter is designed in the context of a real-time 4K HW ASIC VVC decoder. In this case, the 2-D MTS module should sustain a fixed throughput of 1 sample/cycle. Furthermore, to facilitate chaining between different block sizes, the latter should sustain a unified latency regardless of block sizes and transform types. Almost, every state of the art study for the early transform HW design considers mostly a multiplier-less approach a.k.a CM instead of using RM. In this work, a study and comparison for two hardware architectures for the VVC MTS is presented. The first one implements the transform using a CM based architecture, while the second one uses a RM based architecture. Finally, experimental and synthesis results are discussed and compared with related works.

Chapter 6 tackles the problem of hardware implementation of the inverse 1D VVC MTS and LFNST targeting 4Kp60 resolution and 450Mhz frequency. Both, the 1-D iMTS and iLFNST should sustain a fixed system latency with a fixed throughput of 4 samples/cycle. Based on our previous study of Chapter 5, the best approach to implement the iMTS is thought an RM-based architecture. It is also the same approach used to implement the iLFNST. In this work, a unified hardware architectures for the VVC iMTS and iLFNST using an RM-based architecture is described. Finally, the experimental setup and the synthesis results of the VVC transform implementations with a comparison with the HEVC transform is presented.

Chapter 7 presents a comparison between a shared design uniting the DCT-2 and its inverse and a shared design uniting the DCT-2 and the LFNST at a throughput of 32 samples/cycle in the context of a 4K real-time VVC ASIC encoder. Firstly, the cost of each shared design separately is studied. Then, a comparison between both solutions with a highlight of the main differences is presented.

Chapter 8 focuses on the background of the ALF tool of VVC. It presents the different filters used by the ALF for the Luma and Chroma components. In addition, it introduces the existing hardware implementations of the ALF.

Chapter 9 presents a novel hardware-architecture for the VVC ALF filter, considering its three filters: 7×7, 5×5 and 3×4 CCALF. This design is based on the last Working Draft and the last VTM. The proposed design sustains a fixed throughput of 2 pixels/cycle with a fixed system latency that enables the processing of 4K video at 60 frames per second in 4:2:2 Chroma sub-sampling. The proposed design is integrated into a professional video decoder chip supporting multiple video coding standards. Firstly, it describes the top level design of the ALF module. Secondly, it shows the different input scanning along with the most optimal scanning order. Thirdly, it describes the memory configurations of the proposed input scanning order. Then, the proposed design for the gradient module and for the core ALF filters are tackled. Finally, it presents the experimental setup and the synthesis results of the ALF along with a comparison to the state-of-the art works.

Chapter 10 focuses on the hardware design of the ALF in the encoder context. Firstly, it presents the hardware designs of the memory, gradient sum and filtering modules for the Luma ALF for a throughput of two pixels/cycle. Secondly, it describes the ALF decision architecture for a throughput of one pixel/cycle. Thirdly, it presents the hardware designs of the memory, gradient, filtering and the SSD modules for the ALF decision. Finally, it presents the experimental setup and the synthesis results in a comparison with the ALF decoder.

Finally, Appendix A lists the publications produced during the thesis

Part I

Video compression background and real time video codecs

Chapter 3

Video compression background

3.1 Introduction

This chapter introduces the video compression background and its standardization history. Section 3.2 describes some fundamental concepts for video applications such as color spaces, different coding structures and the Common Test Conditions (CTC) applied to evaluate and compare video coding contributions along with the most important used metrics in video coding applications. Section 3.3 is dedicated to describe the history of video compression. Section 3.4 presents a brief overview on the state-of-the-art video coding standard, Versatile Video Coding (VVC) explaining his coding scheme and the main blocks of the coding chain. Section 3.5 presents the characterization of a real time video codec with a comparison between several commercial encoders. Finally, Section 3.6 summarizes this chapter.

3.2 Video Formats and applications

Digital video consists of a succession of images at a certain display speed called frame rate. It is known that 25 frames per second (fps) is the minimum frame rate to trick the human eye and simulate an animated video. In addition, digital video adopts the characteristics of images such as color space, resolution, and bitdepth. Since the digital image is sampled along both spatial axes, it can also be represented by a grid of elementary points called pixels.

3.2.1 Color space and chroma sub-sampling

To represent an image in the RGB space (Red, Green, Blue), each pixel consists of three components: red, green and blue. The addition of these 3 monochromatic colors allows to obtain all the other colors. Other color spaces exist such as YUV (luminance, blue chrominance, red chrominance) which exploits the sensibility of the eye to the light more than the colors. The conversion from RGB space

to YUV space is applied using a linear transformation given by the equation (3.1).

$$\begin{aligned} Y &= 0.299 \times R + 0.587 \times G + 0.114 \times B, \\ U &= -0.169 \times R - 0.331 \times G - 0.5 \times B, \\ V &= 0.5 \times R - 0.419 \times G - 0.081 \times B \end{aligned} \tag{3.1}$$

Therefore, each frame is defined by the three images having the components Y, U and V. In fact, the luminance component Y refers to the gray level of a pixel while the two chrominance components U and V define the colors. As the chrominance has only a weak impact on the human visual system (SVH), it is interesting to reduce the resolution of the color images. This obtained by sub-sampling the colors images while preserving only half of the color information ($4:2:2$) or only a quarter of the color information ($4:2:0$) as shown in Figure 3.1. YUV space is also the adequate space for video compression due to the chroma subsampling.

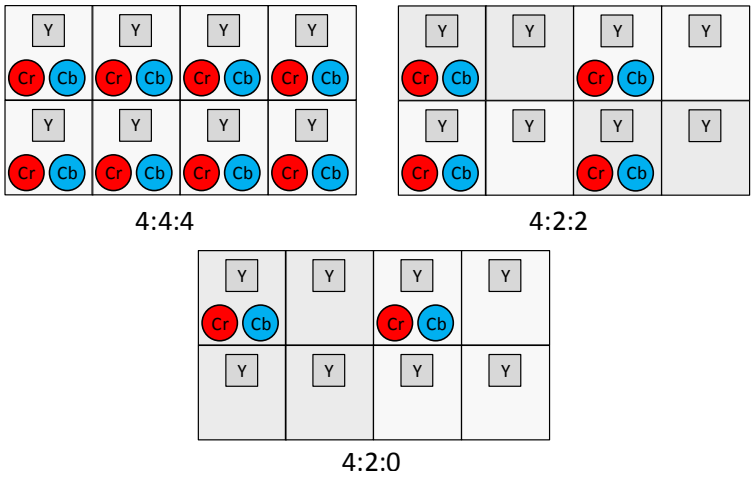


Fig. 3.1 Different Chroma sub-sampling for YCrCb color space

- 4:4:4: Each of the two chrominance arrays has the same height and width as the luma array.
- 4:2:2: Each of the two chrominance arrays has the same height and half the width of the luma array.
- 4:2:0: Each of the two chrominance arrays has half the height and half the width of the luma array.

3.2.2 Structure and configurations of a digital video

We know that a video is a sequence of frames, but these images are actually packaged into a set of Group of Pictures (GoPs), which makes the video as sequence of GoPs as shown in Figure 3.2. The frames of the GOP could be one of three types: I (for Intra), P (for Predicted) or B (for Bi-directional predicted) frames.

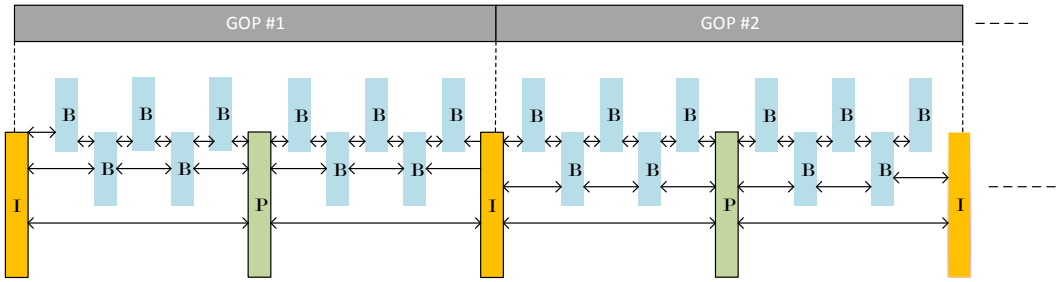


Fig. 3.2 Group of Picture sequencing for RA structure

For video application there are mainly four coding configurations:

- All Intra (AI): In this configuration, all the GOPs are formed by I-frames. Therefore, there is no time dependency between frames and it is less complex than other configurations because it avoids Motion Estimation (ME). This configuration allows fast coding but at low compression rates.

- Random Access (RA): In this configuration, the GOPs are formed from a hierarchical structure of B frames as shown in the example of Figure 3.2. The latter allows the RA configuration to have the highest coding efficiency compared to the others. However, it generates a bit-stream with higher latency due to the reorganization of frames during coding.

- Low Delay P (LD-P): Two types of frames are used in the LD-P configuration: I and P. The first frame of a GOP is an I frame followed by a large number of P frames. Unlike the RA configuration, the frames are not rearranged and the coding order is identical to the display order. This configuration has a higher efficiency than the AI structure and a lower delay than the RA configuration.

- Low Delay B (LD-B): As with the LD-P configuration, the first frame is coded as an I-frame followed by B-frames and the coding order remains the same. In addition, the LD-B configuration achieves better coding efficiency than LD-P due to bi-prediction and provides low delay coding.However, the coding process may be more complex for each frame due to the additional coding options.

3.2.3 Common Test Conditions

CTC are desirable to conduct experiments in a well-defined environment and ease the comparison between video coding contributions. It is a set of requirements to be respected defined by the standardization committee. In the context of VVC, JVET team defined a set of CTC for different video formats and coding configurations, for instance Beson et al. in [12] defined the set rules to be met for SDR videos, Segall et al. in [13] for HDR videos, Hanhart et al. in [14] for 360° videos, Chao et al. in [15] for non-4:2:0 color formats, Nguyen et al. in [16] for lossy/lossless coding configurations and finally Browne et al. in [17] for high bit depth and high bit rates video coding. For each of these cases,

a number of Sequences spread into different classes are defined. Each class represents a particular resolution. For instance, in VVC and for for SDR sequences [12], there are seven fixed classes (A1, A2, B, C, D, E and F). For each sequence of these classes, the resolution, framerate, bitdepth and the number of frames to encode are predefined. The Intra Period, i.e. the frequency of I-frames is also fixed and defined as a function of the framerate.

3.2.4 Distortion measures and optimization

Source coding systems often use lossy coding techniques to achieve a higher compression ratio. These techniques introduces a measurable difference between the signal before coding, known as the source signal, and the signal after coding and decoding, known as the reconstructed signal. The measurement of this difference is called distortion and is noted D. In order to qualify the effectiveness of video coding, metrics for objective and subjective evaluations are used. Subjective metrics qualify a video based on the ratings given by a set of human testers. They are also used to define psycho-visual tools that qualify video coding based on human observations. On the other hand, objective metrics qualify a video based on the measurement of the distortion between the source signal and the reconstructed. Among the most commonly used objective metrics are the followings:

- PSNR: The most widely used and well-known objective measure of distortion is the Peak Signal-to-Noise Ratio (PSNR). It is based on the Mean Squared Error (MSE), which is one of the most used distortion metric in the area of source coding. Its wide adoption into the coding community is due to the low computational cost and the ease of use in mathematical problems. For a source signal O and a reconstructed signal $\hat{O}$ both composed of $M \times N$ samples, the MSE is defined by Equation (3.2), with $O_{i,j}$ being the value of O at coordinate (i, j).

$$MSE(O,\hat{O}) = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(O_{i,j} - \hat{O}_{i,j})^2 \tag{3.2}$$

  The PSNR is calculated from the MSE using Equation (3.3) where $2^B - 1$ represents the maximum value of a sample, with B being the number of bits per sample (bitdepth). The PSNR is expressed on a logarithmic scale (dB) to account for the wide range of possible signals. The higher the value, the better the quality. The PSNR can be calculated on an image or a video sequence.

$$PSNR = 20\log\frac{2^B - 1}{\sqrt{MSE}} \tag{3.3}$$

- SSIM: The Structural SIMilarity (SSIM) metric measures the distortion between two images using the peak-to-peak signal-to-noise ratio [18]. In fact, it measures the difference by structural zones and not by pixel, unlike the PSNR. The SSIM formula is given in Equation (3.4). It is computed on a windows-basis for two signals O (source) and $\hat{O}$ (reconstructed). where μ_O is

the average of O, $\mu_{\hat{O}}$ is the average of $\hat{O}$, σ_O is the variance of O, $\sigma_{\hat{O}}$ is variance of $\hat{O}$, $\sigma_{O,\hat{O}}$ is the covariance of O and $\hat{O}$, $c1$ and $c2$ are two stabilization variables given by Equation (3.5) with B the number of bits per sample and k_1 and k_2 two coefficients (set at 0.01 and 0.03 by default).

$$SSIM(O,R) = \frac{(2\mu_O\mu_{\hat{O}} + c1)(2\sigma_{O,\hat{O}} + c2)}{(\mu_O^2 + \mu_{\hat{O}}^2 + c1)(\sigma_o^2 + \sigma_{\hat{O}}^2 + c2)} \tag{3.4}$$

$$c1 = (k_1 2^B - 1)^2, \;\; c2 = (k_2 2^B - 1)^2 \tag{3.5}$$

It is believed to be more related to perceived quality than PSNR, because the human eye is attracted to specific areas of the screen, such as faces, and would neglect other areas that may have a lower PSNR. The SSIM is a value between 0 and 1 indicating the correlation to the source. The closer the SSIM is to 1, the better the correlation.

- Rate distortion optimazation: The Rate Distortion Optimazation (RDO) tries to answer a simple question: "What part of the video signal should be coded using what method and parameter settings?". The optimization task is to choose, for each image region, the most efficient coded representation in the rate-distortion sense. This problem can be modeled by Lagrangian method using Equation (3.6):

$$J_\lambda = D + \lambda R \tag{3.6}$$

Where J_λ is the RD-cost that refers to the trade-off between the distortion D and the rate R and λ is the Lagrange weighting factor for the cost control. The RDO process exhaustively tests all possible coding parameters and selects those that minimize the RD-cost and finally generates them in the bit-stream. That is why it is considered to be the main source of complexity in the video encoding process.

- Bjontegaard measures: The Bjontegaard metric calculation involves two parts – BD PSNR and BD Rate. It was introduced and explained in [19] and has since provided an efficient comparison between the RD performance of two different codecs or two different settings for the same codec. They measure the average difference between two Rate-Distortion (RD) curves interpolated through multiple bit-rate points. The BD PSNR measures the average PSNR difference in decibels (dB) for two different encoding algorithms considering the same bit-rate. On the other hand, the BD Rate reports the average bit-rate difference in percent for two encoding at the same quality.

3.3 Video Compression history

Historically, video was stored as an analog signal on magnetic tape. Around the time that the compact disc entered the market as a digital format replacing analog audio, it became possible to store and transmit videos in digital form. Because of the large amount of storage and bandwidth required to record and transmit raw videos, a method was needed to reduce the amount of data used to represent the video in a coded way. Since then, engineers and mathematicians have developed a number of solutions to achieve this goal, which involves the compression of digital video data under the scope of finding the most optimal video codec.

A video codec is a software or hardware devise that compresses and decompresses digital videos. In the context of video compression, codec is an umbrella for encoder and decoder, while a device that only compresses is usually called an encoder, and one that only decompresses is a decoder. The format of the compressed data usually conforms to a standard video coding format. This latter defines the bit-stream format and the decoding/decompressing phase while the encoding phase is not standardized and could be defined by the constructor depending on the target application. The compression is usually lossy, which means that the compressed video is stripped of some of the information present in the original video. As a result, the decompressed video has lower quality than the original uncompressed video because there is insufficient information to accurately reconstruct the original video. The most active organizations at the global level in the standardization of video compression systems are ISO-IEC and ITU-T. ISO-IEC's technical work is carried out within the Motion Picture Experts Group (MPEG), which has established the MPEG-1, MPEG-2 and MPEG-4 standards. Parallel to MPEG activities, the ITU-T Video Coding Experts Group (VCEG) is particularly involved in defining technical recommendations for video conferencing and video telephony applications, and has been developing the H.26x standards since 1990 as shown in Figure 3.3.

The H.261 standard was the first video standard approved in 1990 [20]. It was used for video tele-

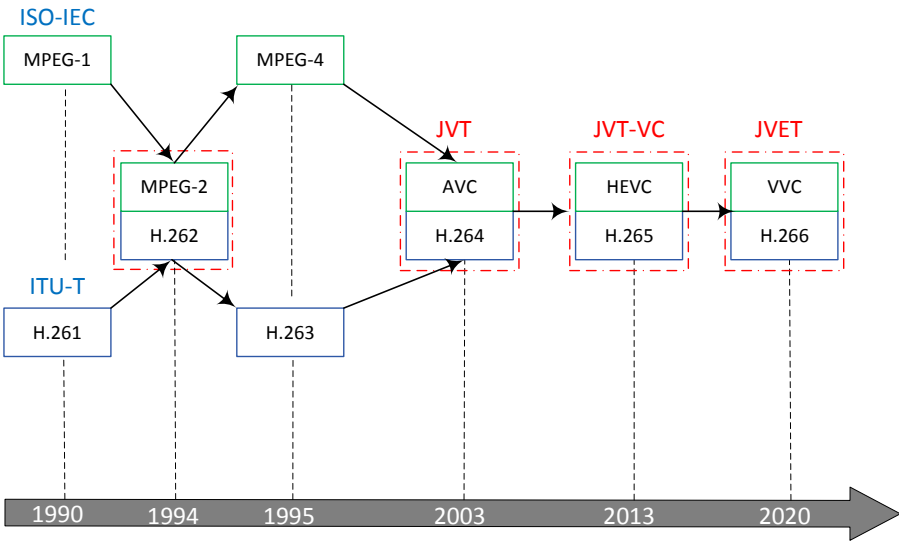


Fig. 3.3 Milestones of video coding standardization history

phony applications. The supported image formats are CIF (288×352 pixels) and QCIF (144 × 176 pixels). The MPEG-2/H.262 standard [21,22] was developed and approved jointly in 1993 by the ISO and ITU standards organizations through the MPEG and VCEG groups. This standard has met with success and mass adoption by the DVD, broadcast/broadband industry such as DTV. This joint collaboration has not prevented these two groups from developing their own standards. H.263 is a video coding standard for ultra-low bit rate video communication, adopted in 1995 [23]. It was intended for video conferencing and video telephony applications. The H.263 standard was then modified to give rise to two new versions, respectively called H.263+ finalized in 1998 and H.263++ finalized in 2000. The latter improved the compression efficiency by 15% to 25% over the first version and supported custom and flexible video formats. Similarly, MPEG has independently developed the MPEG-4 standard [24] containing the basic ITU-T H.263. It addresses a wide variety of audiovisual applications ranging from video conferencing to audiovisual production. Considering traditional video coding, MPEG-4 combines the tools of MPEG-2 with new features resulting in a more efficient and error-resistant compression. The ITU-T and ISO created the Joint Video Team (JVT) in 2001 to develop and finalize a more efficient standard than MPEG-2, called Advanced Video Coding (AVC) [25]. The H.264 / AVC standard, still known as MPEG-4 Part 10, was first approved in May 2003 and is the most widely used video coding standard, especially by telephone companies and Internet video providers. The H.264 / MPEG-4 AVC standard reduces the storage rate by half for the same visual quality compared to previous standards. In January 2005, the MPEG and VCEG groups agreed to finalize the Scalable Video Coding (SVC) project as an extension to the H.264 / MPEG-4 AVC standard.

Over the years, the H.264 / AVC standard has addressed the application needs of multimedia systems, offering superior performance to previous generation video encoders. However, its efficiency is obviously insufficient for the storage and delivery of new video formats at higher resolutions (QFHD and beyond). Subsequently, in 2010, the Joint Collaborative Team on Video Coding (JCT-VC) group, bringing together both MPEG and VCEG experts, was formed with a mission to reduce the bitrate by 50% for the same visual quality compared to H.264/AVC. In January 2013, the famous High Efficiency Video Coding (HEVC) / H.265 codec [26] was finalized allowing double the bitrate for the same subjective video quality compared to H.264/AVC. In 2014, MPEG announced that the second edition of HEVC will be accompanied by three extensions, namely multiview extensions (MV-HEVC), range extensions (REext), and the scalable HEVC extension (SHVC).

Finally, the Joint Video Expert Team (JVET), created in 2015 and bringing together experts from MPEG and VCEG, finalized a new video codec and a successor to the HEVC standard [27]. This new video coding standard is called Versatile Video Coding (VVC) and was published in July 2020 [28, 29]. In the same way, VVC allows for a 50% reduction in bit rate for the same subjective video quality compared to the HEVC standard [30, 31]. It also supports a wide range of applications beyond the typical standard and high-definition camera-captured content codings, including features to support computer-generated/screen content, high dynamic range content, multi-layer and multi-view coding, and support for immersive media such as 360° video.

3.4 Versatile Video Coding (VVC)/H.266

The main goal of video coding tools is to remove redundancy from multimedia data. The main types of redundancies that video coding aims to remove are:

- *Spatial redundancy*: Refers to the high correlation between neighboring pixels in an image. This redundancy may be seen specially for pixels in regular objects or background areas where the distribution of brightness and color saturation are regular.
- *Temporal redundancy*: Indicates the high correlation among pixels of neighboring frames in a video sequence. This correlation appears since the physical characteristics between neighboring frames are really similar, as they where captured in small time intervals.
- *Statistical redundancy*: This redundancy refers to the coding process. Symbols of a bit stream do not appear with the same probability in the real world. Thus, the encoder must try to code the most frequent symbols with a lower number of bits to reduce the size of the bit-stream.
- *Visual redundancy*: Refers to those details in video images not easily perceived by the Human Visual System (HVS).

The video coding scheme is actually called hybrid due to the combination of temporal and spacial predictions between the blocks of the video sequence and the trigonometric transformation of residuals (the difference between the predicted and the original blocks). Figure 3.4 shows a simplified structure of the hybrid video coding scheme.

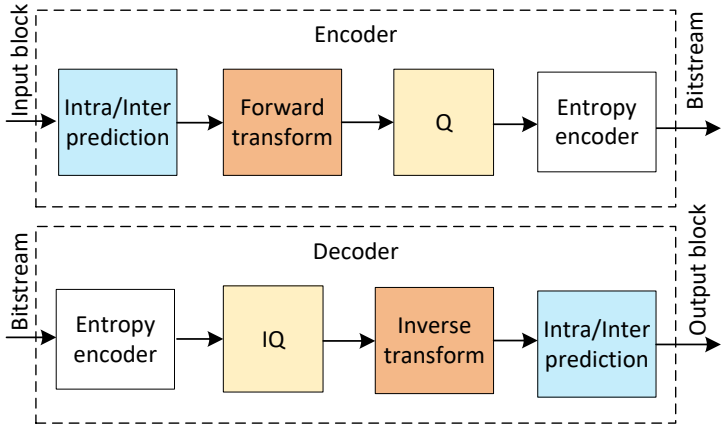


Fig. 3.4 Structure of the hybrid video coding schema

Inter/Intra prediction, transformation, quantization and entropy coding are the basic processing steps in the video encoder to produce a compressed bitstream. Then, the bitstearm will be the input of the decoder chain in order to regenerate the image. This is through the entropy decoding, inverse quantization, inverse transform and finally the Inter/Intra prediction.
VVC [29] is a block based hybrid video coding scheme such as its predecessor HEVC [9]. The VVC

encoder starts by dividing the image into block-like regions called Coding Tree Unit (CTU)s with the maximum allowed CTU size is 128×128. Then, using Quad-Tree (QT) [32] and Multi Type Tree (MTT) procedures [33], a further partitioning is applied to the CTUs. The first image is always intra-coded because it will be used as a reference. Otherwise, a predicted signal is generated for each sub-block, using either intra-frame prediction or inter-frame prediction, called Motion-Compensated Prediction (MCP).

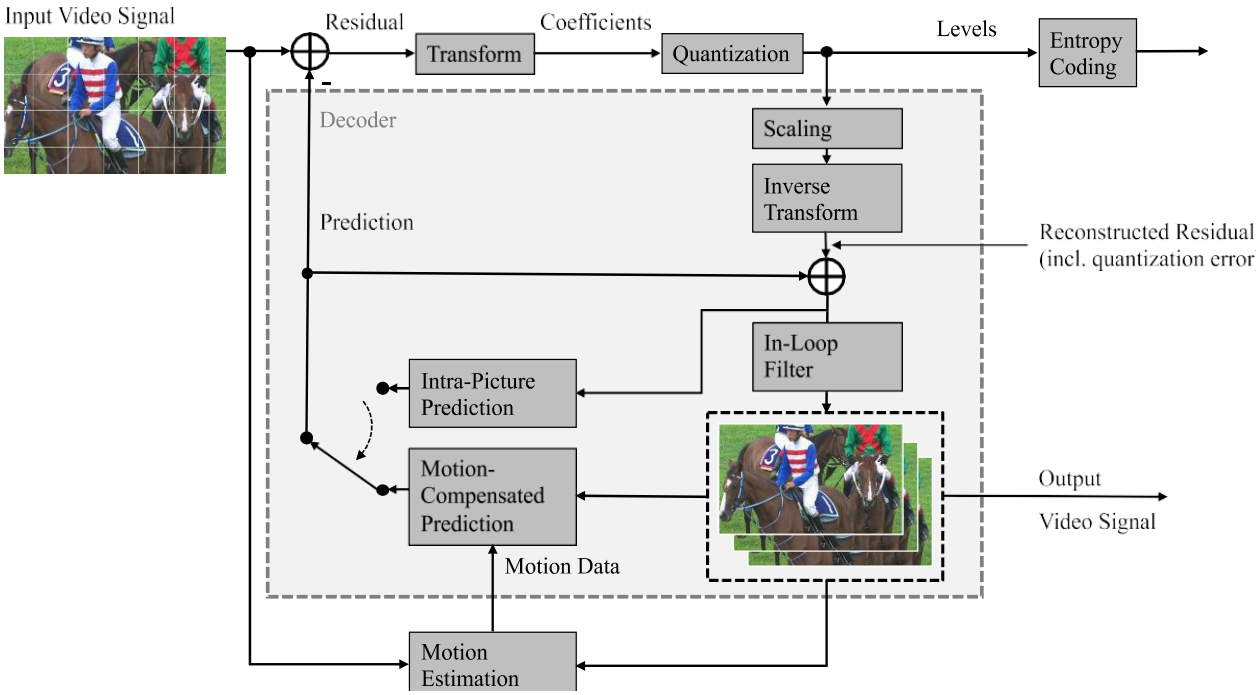


Fig. 3.5 Structure of the VVC video coding scheme

Figure 3.5 shows the hybrid structure of the VVC video encoding schema. The encoder uses a decoding algorithm (shown by the gray box) to reconstruct the compressed images for use in temporal prediction. To do this, an inverse quantization and an inverse transformation are applied to reconstruct the residuals. Then, the residual is added to the predicted samples to generate the decoded frame. The result goes through the filtering operations: four loop filters are used for VVC to smooth the distortions at the block boundaries and to reduce the error. The reconstructed frames are then stored in a reference buffer used for inter-coding of other images. In this section, we aim to go through every part of the VVC encoding process in their execution order.

3.4.1 Block Partitioning

The first step of the video encoding is frame partitioning. Every frame is divided into a sequence of CTUs. For an image with three sample arrays, a CTU consists of an N×N block of luma samples and two corresponding blocks of chroma samples. Figure 3.6 shows the example of a picture divided into CTUs. The maximum allowed Luma block size in a CTU is specified as 128×128 (although the maximum Luma transform block size is 64×64). A CTU is further split into Coding Units (CU)

by using a Quaternary-tree with nested multi-type tree. The decision whether to code a picture area using inter-picture (temporal) or intra-picture (spatial) prediction is made at the level of CU. In the coding tree structure, a CU can have either a square or rectangular shape.

A CTU is first partitioned by a Quaternary tree (a.k.a. Quadtree) structure. The leaf nodes of the

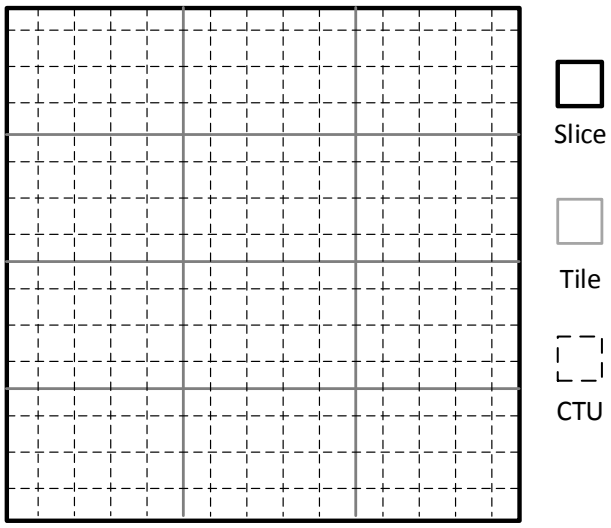


Fig. 3.6 Example of a picture divided into CTUs, Slices and Tiles

Quaternary tree can then be subdivided by a multi-type tree structure. As shown in Figure 3.7, there are four types of splitting in the multi-type tree structure: vertical binary splitting (SPLIT_BT_VER), horizontal binary splitting (SPLIT_BT_HOR), vertical ternary splitting (SPLIT_TT_VER), horizontal ternary splitting (SPLIT_TT_HOR) and Quadtree split (SPLIT_QT). The nodes of the leaves of the multi-type tree are called coding units (CUs), and unless the CU is too large for the maximum length of the transform, this segmentation is used for prediction and processing of the transform without further partitioning.

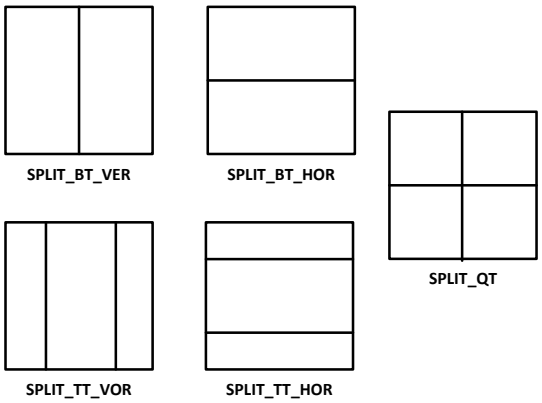


Fig. 3.7 Multi-type tree splitting modes

Figure 3.8:(A) illustrates the mechanism for partitioned CTU in a multi-type nested Quaternary coding tree structure. A CTU is treated as the root of a Quadtree and is first partitioned. Each leaf node of the Quadtree (when large enough to allow it) is then subdivided by a multi-type tree structure. Figure 3.8:(B) shows a CTU example divided into multiple CUs with a quadtree and nested multi-type tree coding block structure, where the bold block edges represent quadtree partitioning and the remaining edges represent multi-type tree partitioning. The quadtree with nested multi-type tree partition provides a content-adaptive coding tree structure comprised of CUs. The size of the CU may be as large as the CTU or as small as 4×4 in units of luma samples. However, for Intra coded blocks, CUs can have a maximum size of 64×64. For the case of the 4:2:0 chroma format, the maximum chroma CB size is 64×64 and the minimum chroma CB size is 2×2. In VVC, the maximum supported luma transform size is 64×64 and the maximum supported chroma transform size is 32×32. When the width or height of the CB is larger than the maximum transform width or height, the CB is automatically split in the horizontal and/or vertical direction to meet the transform size restriction in that direction.

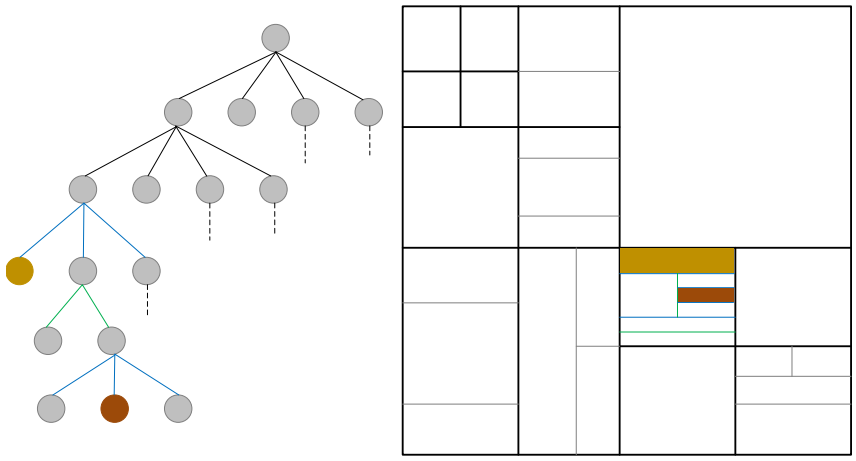


Fig. 3.8 Example of Quad-tree with nested multi-type tree coding block structure

3.4.2 Intra-Picture Prediction

Intra prediction, sometimes called spatial prediction, consists in reducing spatial redundancies in the frame. The basic principle of intra prediction is that the texture of a region of the image is similar to the texture of its neighborhood and can be predicted from there. To capture the arbitrary edge directions presented in natural video, the number of directional intra modes in VVC is extended from 33, as used in HEVC [9], to 65. Because VVC introduces the use of rectangular blocks, several conventional angular intra prediction modes are adaptively replaced with wide-angle intra prediction modes for the non-square blocks. For directional modes, the pixels of the predicted block are interpolated at a defined angle, as shown in Figure 3.9. The intra predictor can be coded in two ways. The first one is the Most Probable Modes (MPM) method which uses the coding modes of the surrounding

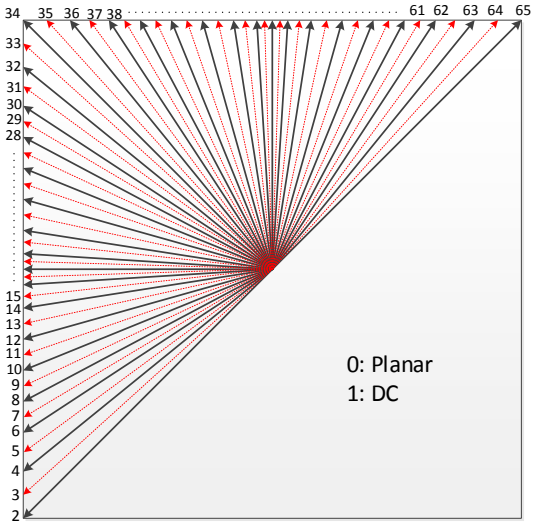


Fig. 3.9 Intra prediction modes

CUs, i.e. left and top. When the considered intra predictor matches one of the predictors included in the MPMs, only the index in this reduced set is transmitted. The second option, where MPMs are not used, is to use a fixed length code. In addition, the precision of the interpolation filters used to generate the intra prediction block with angular prediction modes is increased by using 4-tap instead of 2-tap filters (for HEVC) to improve boundary smoothing with previous reconstructed samples. The results of intra prediction of planar mode are further modified by a position dependent intra prediction combination (PDPC) method. PDPC is an intra prediction method which invokes a combination of the un-filtered boundary reference samples and HEVC style intra prediction with filtered boundary reference samples. To reduce the cross-component redundancy, a Cross-Component Linear Model (CCLM) prediction mode is used, for which the chroma samples are predicted based on the reconstructed luma samples of the same CU by using a linear model. VVC introduces a set of new features dedicated to intra prediction [34] and they are the following:

- Multiple reference line (MRL) intra prediction.
- Intra sub-partitions (ISP) enabling 1-D intra prediction and transform blocks.
- Matrix weighted Intra Prediction (MIP).

3.4.3 Inter-Picture Prediction

Inter prediction reduces temporal redundant information using motion compensation (MC), which relies on the fact that the difference between adjacent frames of the video sequence results from camera and object movements. On the encoder side, the motion estimation (ME) step is performed. ME searches for the best matching area in the reference image for the current prediction block. This is one of the most complex parts of video encoders in terms of computational requirements. Once a good prediction has been found, a motion vector is computed, indicating the offset that should be

applied in the block from the reference image. For VVC standard (VTM), in his last draft includes a set of new inter prediction coding tools [33]:

- *Extended merge prediction*: For each CU, an index of best merge candidate is encoded using truncated unary binarization (TU).
- *Affine motion compensated prediction*: This mode can be applied for CUs with both width and height larger than or equal to 8. In this mode, the Control Point Motion Vectors (CPMVs) of the current CU is generated based on the motion information of the spatial neighboring CUs. There can be up to five CPMV candidates and an index is signalled to indicate the one to be used for the current CU.
- *Adaptive motion vector resolution (AMVR)*: In VVC, a CU-level adaptive motion vector resolution (AMVR) scheme is introduced. AMVR allows MVD of the CU to be coded in different precision.
- *Motion field storage*: In VVC, the highest precision of explicitly signalled motion vectors is quarter-luma-sample. In some inter prediction modes such as the affine mode, motion vectors are derived at 1/16th-luma-sample precision and motion compensated prediction is performed at 1/16th-sample-precision. In terms of internal motion field storage, all motion vectors are stored at 1/16th-luma-sample precision.
- *Bi-directional optical flow (BDOF)*: The bi-directional optical flow (BDOF) tool is included in VVC. BDOF, previously referred to as BIO, was included in the JEM. Compared to the JEM version, the BDOF in VVC is a simpler version that requires much less computation, especially in terms of number of multiplications and the size of the multiplier. BDOF is used to refine the bi-prediction signal of a CU at the 4×4 subblock level.
- *Decoder side motion vector refinement (DMVR)*: In VVC, the DMVR can be applied for the CUs which are coded with multiple modes and features, in which we site: CU level merge mode with bi-prediction MV, CU has more than 64 luma samples, BCW weight index indicates equal weight.
- *Combined inter and intra prediction (CIIP)*: In VVC, when a CU is coded in merge mode, if the CU contains at least 64 luma samples (that is, CU width times CU height is equal to or larger than 64), and if both CU width and CU height are less than 128 luma samples, an additional flag is signalled to indicate if the combined inter/intra prediction (CIIP) mode is applied to the current CU. As its name indicates, the CIIP prediction combines an inter prediction signal with an intra prediction signal.

3.4.4 Transform and Quantization

The objective of transform domains is to concentrate as much as possible the residual signal in the top left corner (low frequencies) carrying the most significant information. This idea of the energy compaction is the main property of the transform stage. Most of the transforms used in standardized

video coding schemes belong to the Discrete Trigonometric Transform (DTT) family. The VVC transform includes three transform tools:

- *Multiple Transform Selection (MTS)*: The MTS tool involves three trigonometrical transform types including Discrete Cosine Transform (DCT) type 2 (DCT-2), DCT type 8 (DCT-8) and Discrete Sine Transform (DST) type 7 (DST-7) with block sizes that can reach up 64×64 for DCT-2 and 32×32 for DCT-8 and DST-7 [35]. Chapter 4: section 4.2 is completely dedicated to give a detailed overview of MTS, so further explanation will be given there.

- *Low Frequency Non Separable Transform (LFNST)*: The LFNST relies on matrix multiplication applied between the forward primary transform (MTS) and the quantisation at the encoder side. It uses a set of predefined matrices coefficients and applied only when the DCT-II is selected for the MTS [36]. Four sets of two LFNST kernels of sizes 16×16 and 48×16 are applied on 16 coefficients of small blocks ($min(width, height) < 8$) and 48 coefficients of larger blocks ($min(width, height) > 4$), respectively. Chapter 4: section 4.3 is completely dedicated to give a detailed overview of LFNST.

- *Sub-block Transform (SBT)*: In VTM, sub-block transform is introduced for an inter-predicted CU. In this transform mode, only a sub-part of the residual block is coded for the CU. When SBT is used, the transform block is either half or quarter size of the residual block. The SBT transform kernel is selected based on the transform block position. Position-dependent transform core selection is applied on luma transform blocks in SBT. The two positions of SBT Horizontal and SBT Vertical are associated with different core transforms. More specifically, the horizontal and vertical transforms for each SBT position is specified as shown in Figure 3.10. For example, the horizontal and vertical transforms for SBT-V position 0 is DCT-VIII and DST-VII, respectively. When one side of the residual TU is greater than 32, the transform for both dimensions is set as DCT-II.

After transformation, the transformed coefficients are quantized using a non-linear discrete mapping of the coefficient values into integer quantization indices reducing the amount of possible output values. The quantization is scalar: each coefficient is approximated independently of its neighboring values. In VVC, the quantization step is controlled by a Quantization Parameter (QP) that discards any coefficient whose energy level is below a certain threshold. In addition, a new concept called dependent scalar quantization is used. Dependent scalar quantization refers to an approach in which the set of admissible reconstruction values for a transform coefficient depends on the values of the transform coefficient levels that precede the current transform coefficient level in reconstruction order. More details can be found in [37, 38].

3.4.5 In-loop Filters

In-loop filters are located in the decoding loop of the encoder. During all the video coding stages, and especially in the lossy compression performed in the quantization stage, the subjective quality of a video sequence can be reduced resulting in the appearance of blocking, ringing or blurring artifacts.

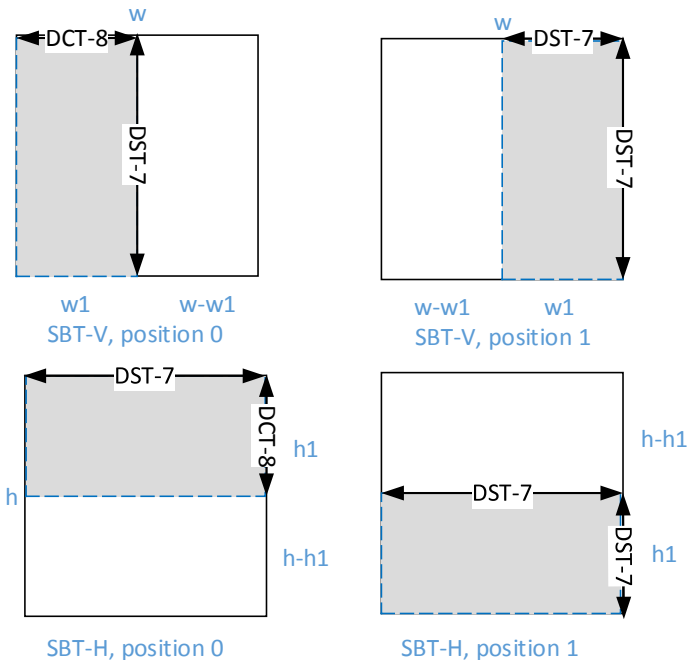


Fig. 3.10 SBT position and transform type

An example of an image showing blocking and ringing artifacts may be seen in Figure 3.11. In order to remove these artifacts, and increase the subjective and objective quality of the reconstructed sequence, a set of in-loop filters are used. In-loop filters in the encoder estimate the optimal filter parameters that increase the objective quality of a frame the most. These parameters are then transmitted to the decoder so that the in loop filters of the decoder can use these parameters to optimally filter the reconstructed frame and achieve the same quality improvements reached for the reconstructed frame in the encoder. VVC uses the following filters for its In-loop filter block [33, 39]:

- *Luma Mapping with Chrom Scaling (LMCS) [40]:* LMCS is a new coding tool only present in VVC. LMCS has two main components: 1) a process for mapping input luma code values to a new set of code values for use inside the coding loop; and 2) a luma-dependent process for scaling chroma residue values. The first process aims at improving the coding efficiency for standard and high dynamic range video signals by making better use of the range of luma code values allowed at a specified bit depth. The second process manages relative compression efficiency for the luma and chroma components of the video signal. The luma mapping process of LMCS is applied at the pixel sample level, and is implemented using a piecewise linear model. The chroma scaling process is applied at the chroma block level, and is implemented using a scaling factor derived from reconstructed neighboring luma samples of the chroma block.

- *Deblocking Filter (DF) [41]:* The purpose of the Deblocking filter is to remove blocking artifacts that appear on the edges of CUs, due to the use of a block structure in encoder processing. The different values between neighboring CUs generate strong edges between CU transitions,

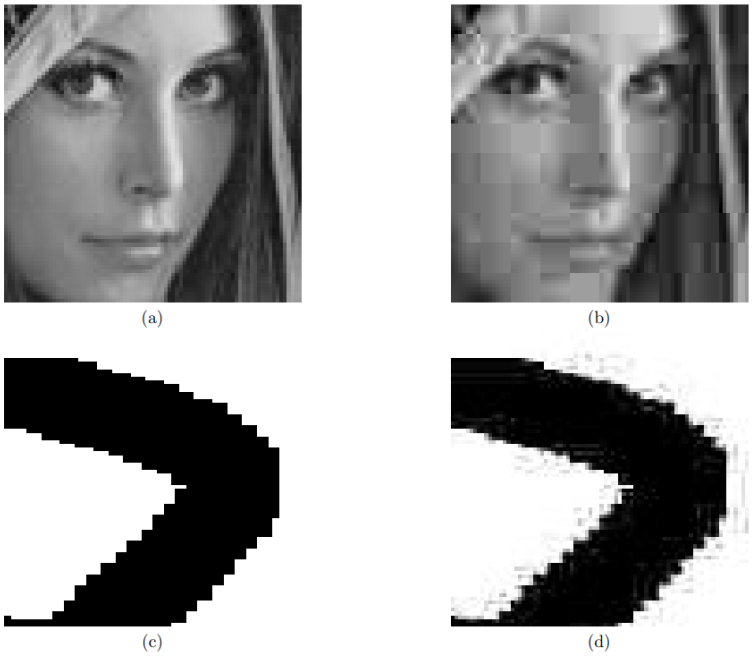


Fig. 3.11 Blocking and ringing artifacts in highly compressed images. (a) and (c) shows portions of original uncompressed images, (b) and (d) shows the same portions compressed. Blocking artifacts are visible in (b) and ringing artifacts are visible in (d)

resulting in these blocking artifacts if there is no smooth filtering on the block edges. Therefore, the deblocking filter will be responsible for applying a smoothing filter to the edges to remove these artifacts. A different type of smoothing filter is applied to a block based on the properties of its neighboring blocks. The strength of the filter will also be determined by the specific pixel values of the block's edges.

- *Sample Adaptive Offset (SAO) [42]:* After removing blocking artifacts, the SAO filter aims to reduce unwanted visible artifacts, such as ringing. The SAO filter classifies the pixels of a CTU into different categories. A positive or negative offset is determined for each category. This offset is then added to all samples in each category. Thus, if the reconstructed samples in a specific category have a smaller value than the original samples, a positive offset is applied to reduce the existing error. On the other hand, if the samples in a specific category have higher values than their corresponding original samples, a negative offset will be applied.

- *Adaptive Loop Filter (ALF): [39]* Adaptation Parameter Set (APS) also aims to reduce visible artifacts such as ringing and blurring by reducing the mean absolute error between the original image and the reconstructed image. Chapter 8 is completely dedicated to give a detailed overview of APS.

3.4.6 Entropy Coding

Entropy coding is the last step in the coding process (the first in the decoding process) that converts the video signal to a series of syntax elements. This latter describes how to reconstruct the input video signal, including the prediction information. In VVC, the entropy coding consists of the following steps; each syntax element is firstly converted into binary symbols. Then, context modeling estimates the probability of each symbol, coded according to its specific context. Finally, whatever the probability, arithmetic coding is used in order to obtain the bit-stream. VVC also uses the Context-adaptive binary arithmetic coding (CABAC) [43] core engine and modified context modeling for transform coefficients as well as transform skip block. More information about the Entropy coding for VVC can be found in [38, 44, 45].

3.5 Characterization of real-time video encoder

3.5.1 Video encoding in today's devices

Video encoding has many application in today's electronic devices. File-based video encoding, is currently the most common. Any video you may watch online has likely been encoded in a data center and stored, ready for delivery to consumers via a service platform such as YouTube, Netflix and other internet applications. Video communication and video calls is also one of the most common application, specially with the growth of live video transcoding (decoding and re-encoding), driven primarily by mobile technology and social media. As consumers become more accustomed to being able to consume content on their mobile devices while on the go and stay connected to family and friends wherever they are, the ability to integrate live videos into these interactions is becoming increasingly desirable. A great example of live video coding in today usage is Facebook diverse feature, which allows a user to engage in a one-to-many broadcast of a live video to their Facebook friends, or the ability to video call family and friends with the possibility of live interaction by sharing reactions, such as a "Like," in real time.

In a live video broadcast over the internet, a single video stream is sent from the source to the cloud. Today, this is achieved purely through software, typically open source encoding projects such as FFmpeg, using many central processing units (CPUs). The difficulty with this approach for live video is that there is a parallelism limit that can be exploited to make the video smaller; this capability is defined by the number of cores within the platform in question. Because the frames per second (fps) must be maintained to avoid delayed playback, the computing requirement must not exceed this FPS at any time. As such, the highest quality settings in a software based codec cannot be reached. For real time videos, software-based coding is simply unable to achieve the maximum quality offered by the encoder standard technology. The fundamental problem with software-based encoding for real time videos is that the best compression - that is, the highest quality video for the lowest bit rate - and the finest end result in video quality are unattainable with the available computation level. This is where the application of real time video encoding is more suitable for dedicated hardware designs. Using hardware platforms such as FPGA or ASIC enables the access for more parallelism capability and higher computational power. An FPGA is basically a type of chip that can be configured by rewriting

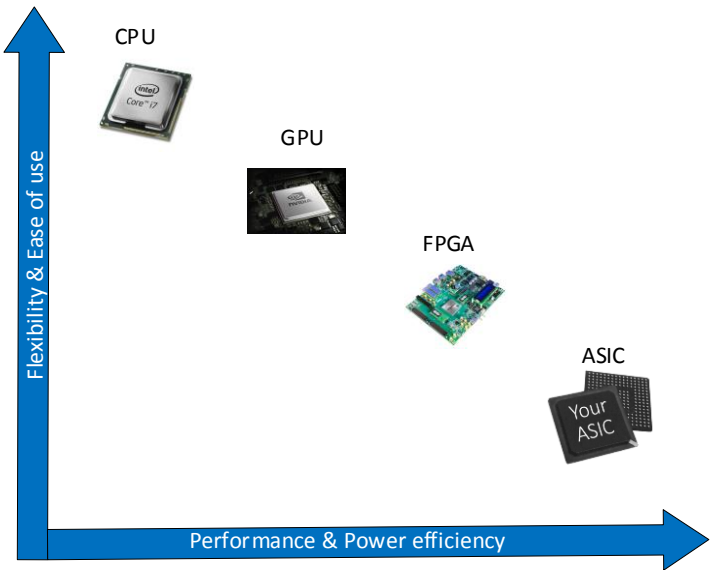


Fig. 3.12 Comparison of chip types & capabilities

its circuit in the field. This is identical to how a traditional ASIC is designed. However, unlike FPGAs, ASICs can not be changed once created, and typically take time to develop from the initial design to deployment. A study conducted by Xilinx [11] showed that the use of dedicated hardware design through FPGA or ASIC for real time video coding is more efficient in term of achievable quality and power consumption. Figure 3.12 shows a comparison of different platforms used for real time video coding and their respective performance.

Thanks to how ASICs are programmed versus running software on CPUs and GPUs, ASICs offer twenty times more performance and consumes around twenty times lower power than CPUs. The main drawback is that they are more difficult to design and use and intended only for one specific application.

3.5.2 Real time encoders

Video coding standards are instantiated in software or hardware with goals as varied as research, broadcasting, or streaming. A 'reference encoder' is a software implementation exploited during the video standardization process. It is used primarily for research purposes, and also as a reference for the implementation process. Generally, this is the first implementation of a standard that is not intended for production. Afterward, encoders developed by the open-source community or commercial entities come along. These are practical implementations deployed by most companies for their encoding needs and are subject to stricter speed and resource constraints. Therefore, the performance of reference and production encoders might be substantially different. Also, encoders may differ between one and the other, in term of performance (energy consumption, run-time, ...) and compression ratio, depending on the target market. In order to evaluate the suitability of a video encoders for specific applications, the following criteria are used:

- **Video quality:** Assessing the quality of video encoded at a target bit rate is an important part of benchmarking a video encoder. Quality assessment can be accomplished using objective quality measurements, such as SSIM or PSNR, and by capturing an observer's subjective experience using subjective tests.

- **Latency:** Another important factor in characterizing a real-time encoder is the latency between video capture and broadcast.

- **Size Weigh and Power (SWaP):** Power consumption is always a critical aspect of an encoder, whether it is a server-based or a portable encoder. The best SWaP is composed of the best trade-off between four main aspects: size, weight and power consumption.

- **Capability score:** is measured by the features offered by the encoders such as support for resolutions, standards, profiles, audio channels, etc.

A successful real-time HW encoder must solve the above problems with multi-variable analysis. Ultimately, this exercise leads to carefully selecting and adapting coding tools for a video standard in terms of complexity and coding gain to meet the application and design requirements.

At this date, very few commercial encoder comparisons have been published, likely attributed to the high cost of conducting needed tests, as well as the difficulty of defining parameters that can be easily reproduced. Among the limited studies in this category, the annual Video Codec Comparison of Moscow State University (MSE) [1]. The comparison provides both objective (SSIM, PSNR and VMAF) and subjective scores, as well as bitrate distance from the target command. In 2018, the MSE launched a large-scale comparison for commercial software-based encoders. This was also the first year that MSE included subjective evaluation in addition to objective measures. Figure 3.13, shows the result of the 2018 MSE comparison.

The performance graph shows that x265 real-time codec was ranked first in term of bit-rate saving, near to x264 and sz265 encoders while being compared to offline encoders compressing high quality content. According to this study, the x264 encoding core, a real-time codec, has the lowest relative encoding time while the x265 encoding core has the lowest compression ratio compared to other software encoders running below 10 fps on very powerful CPUs. In another internal study conducted at VITEC, several commercial encoders were tested for their practical aspects, objective quality, latency, SWaP characteristics, and capability score, which is measured according to features offered by encoders. Figure 3.14 shows a graph representing the above characteristics to define a commercial encoder's overall suitability for specific applications.

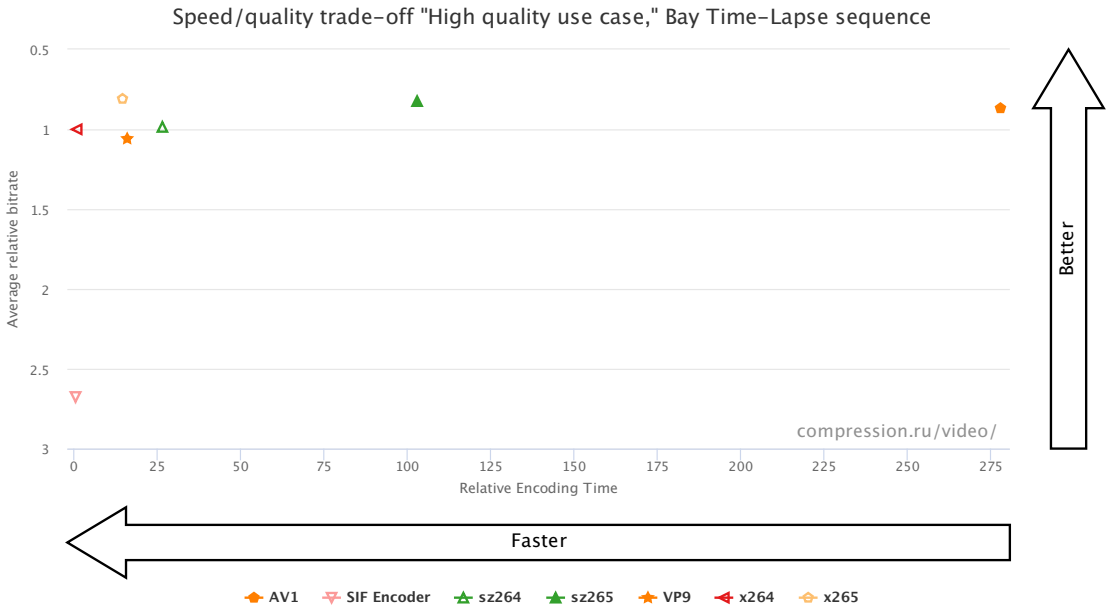


Fig. 3.13 Subjective Quality/Speed comparison, MSU 2018 [1]

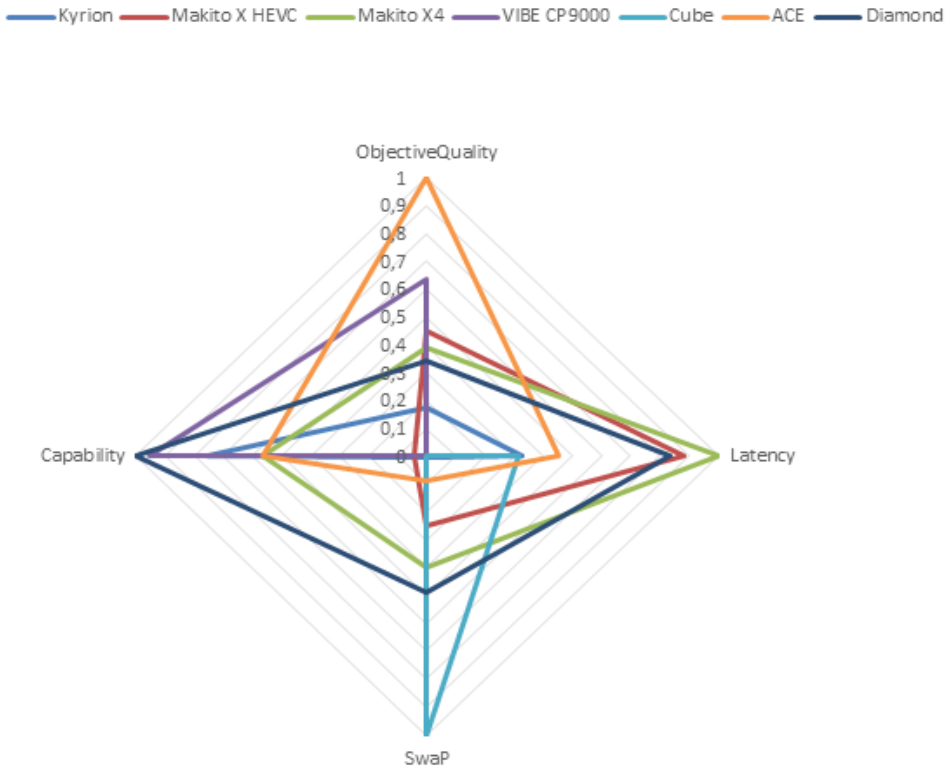


Fig. 3.14 Overall evaluation for 7 point-to-point commercially encoders with objective quality, latency, Capability score, and SWaP characteristics (internal VITEC study).

The comparison of real-life products, especially hardware-based systems, remains a private, behind-the-scenes activity executed by end-users of professional encoders, using their own methodologies. However, this kind of assessment can be of interest for the video community with the benefit of identifying differences between compression quality when based on performance results for a specific target market or product requirement.

3.6 Conclusion

This Chapter presented a general overview of video compression environment. In the last three decades, ISO and ITU-T are the most known standardization organizations starting from the development of H.261 and MPEG-1 until the latest H.266/VVC standard. The format of video sequences and history of compression and fundamental notions for video applications are described in the first part of this chapter. Indeed, ISO and ITU-T have joined their forces to develop standards through the years until the VVC release as the latest official video coding standard. It presented the principles of video coding and the background of the VVC with an emphasis on its specifications characterizing the basic processing blocks of the encoding chain including block partitioning, inter/intra prediction, transformation, quantization, In-loop filtering and entropy coding. Moreover, this chapter presented a bookmarking of real time hardware encoders and their performances and characteristics. In the next parts, the Transform domain of VVC is addressed, we give an overview of the background of the MTS and LFNST tools and a detailed description of their proposed hardware designs in the context of both, encoder and decoder.

Part II

Lightweight hardware implementation of VVC Transform for ASIC platform

Chapter 4

Background and related hardware design studies of the VVC Transform

4.1 Introduction

Video codec uses transforms to ensure a compact signal representation. The transform is a major stage in video codecs. Common video coding standards like Advanced Video Coding (AVC) and High Efficiency Video Coding (HEVC) use multiple variety of the Karhunen-Loéve Theorem (KLT) transform approximation to perform the signal compression. The Discrete Cosine Transform (DCT)-2 is one of KLT approximation, it approximates the KLT following the first order of Markova decomposition. Both, HEVC and AVC, supports only square transform blocks, but for transform blocks of size 4×4, HEVC adds another type of the discrete trigonometric transform family which is Discrete Sine Transform (DST)-7. The newly introduced Versatile Video Coding (VVC) standard uses more varieties of the discrete sinusoidal family and supports non-square transform blocks, it introduces the concept of Multiple Transform Selection (MTS) that involves three types of the trigonometric families, DCT-2, DCT-8 and DST-7 and it includes up to 64×64 block size in addition it uses the Low Frequency Non-Separable Transform (LFNST) as a second step of the VVC transform process. The discrete sinusoidal transform family is known as separable transforms. The use of separable transform reduces significantly the complexity of the transformation process. The ability to reuse coefficient reduces the total number of transform coefficients to be stored and it eliminates additions tasks of pre-processing and post-processing, for rearrangement. This comes with the cost of quality degradation for certain image characteristics like texture patterns. In literature there are a bunch of studies that focuses on the side effect of using only a separable transform. [46] and [47] found that using a single separable transform is rather limited for the highly dynamic image statistics like texture and arbitrarily directed edges, as a solution they introduce the use of a non-separable transform alongside the primary transform for a better compression efficiency specifically for images with texture patterns. The non-separable or secondary transform started to be part of the transformation process for the new emerging video coding standard VVC at the JVET meeting of June 2019 [48]. At first it was named the Non-Separable Secondary Transform (NSST), but lately the name was changed to LFNST as the first non-separable transform that is used for video coding and operates only on low

frequency coefficients [35].

The MTS tool involves three trigonometrical transform types including DCT type 2 (DCT-2) , DCT type 8 (DCT-8) and DST type 7 (DST-7) with block sizes that can reach 64×64 for DCT-2 and 32×32 for DCT-8 and DST-7. The use of DCT/DST families gives the ability to apply separable transforms as the transformation of a block can be applied separately in horizontal and vertical directions. Therefore, the VVC encoder selects a combination of horizontal and vertical transforms that minimizes the rate-distortion cost J, as introduced in the previous chapter.

The LFNST is applied after the separable transform and before the quantisation at the encoder side. At the decoder side, it is applied after the inverse quantisation and before the inverse separable transform. Figure 4.1 illustrates the VVC transform module at both encoder and decoder sides. The transform module relies on matrix multiplication with $\mathcal{O}(N^3)$ and $\mathcal{O}(N^4)$ computing complexities for separable and non-separable transforms, respectively. In terms of memory usage, the VVC transform module requires higher memory allocated to store the coefficients of the transform kernels: three kernels are defined for MTS and eight kernels for LFNST.

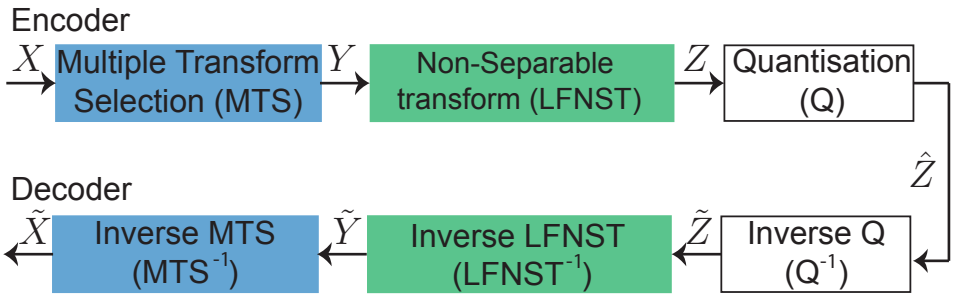


Fig. 4.1 VVC transform module

This chapter explains the concept of each of these tools, and is organized as follows. Section 4.2 presents the background of the separable transform module MTS and section 4.3 details the non-separable transform LFNST in VVC. The existing hardware implementations of the transforms module are presented in Section 4.4. Finally, Section 4.5 concludes the chapter.

4.2 Background of the separable transform (MTS)

The concept of separable 2D transform enables applying two 1D transforms separately in horizontal and vertical directions

$$Y = T_V \cdot X \cdot T_H^T, \tag{4.1}$$

where X is the input residual matrix of size M×N, T_H and T_V are horizontal and vertical transform matrices of sizes N×N and M×M, respectively. "·" stands for matrix multiplication. The inverse 2D separable transform is expressed in (4.2).

$$\tilde{X} = T_V^T \cdot \tilde{Y} \cdot T_H. \tag{4.2}$$

The MTS block in VVC involves three transform types: DCT-2, DCT-8 and DST-7. The DCT-2 is applied for rectangular transform blocks of size $N{\times}M$ with $N,M \in \{4,8,16,32,64\}$ and DST-7/DCT-8 are applied for rectangular blocks of size up to 32×32. The kernels of DCT-2 (C_2), DST-7 (S_7) and DCT-8 (C_8) are derived from (4.3), (4.4) and (4.5), respectively.

$$C^N_{2i,j} = \gamma_i \sqrt{\frac{2}{N}} \cos\left(\frac{\pi(i-1)(2j-1)}{2N}\right), with\ \gamma_i = \begin{cases} \sqrt{\frac{1}{2}} & i=1, \\ 1 & i \in \{2,\ldots,N\}. \end{cases} \tag{4.3}$$

$$S^N_{7i,j} = \sqrt{\frac{4}{2N+1}} \sin\left(\frac{\pi(2i-1)j}{2N+1}\right). \tag{4.4}$$

$$C^N_{8i,j} = \sqrt{\frac{4}{2N+1}} \cos\left(\frac{\pi(2i-1)(2j-1)}{2(2N+1)}\right), \tag{4.5}$$

with $(i,j) \in \{1,2,\ldots,N\}^2$ and N is the transform size.

To avoid floating point operations, all transform kernels are scaled by a pre-defined factor and rounded to the nearest integer. The additional integer transform kernels defined in VVC are derived by scaling the real value of the transform kernel with the factor $64\sqrt{N}$, where N is the transform size, and further adjusted by ±1 after rounding. The MTS transform kernels are defined in way that reduces the complexity of software and hardware implementation.

- **DCT-2:** The elements of the core DCT-2 matrices used in VVC were derived by approximating scaled DCT-2 basis functions according to the Transform Unit (TU) size ranging from 4×4 to 64×64, under considerations such as limiting the necessary dynamic range for transform computation and maximizing the precision and closeness to orthogonality. For simplicity, only one integer matrix for the length of 64 points is specified, and sub-sampled versions are used for other sizes. Although the standard specifies the transform simply in terms of the value of a matrix, the values of the entries in the matrix were selected to have key symmetry properties that enable fast partially factored implementations with far fewer mathematical operations than an ordinary matrix multiplication. With its recursion property, the larger transforms can be constructed by using the smaller transforms as building blocks which is beneficial especially for hardware designs. The adjustment of 64-point DCT-2 is performed in a way that all the DCT-2 kernels defined in HEVC are included, partial butterfly [49] is supported and kernel elements are optimized towards better orthogonality. Butterfly is one of the key features for efficient hardware designs. It allows to reduce the number of multiplications and reduces the memory usage thanks to the fact that each $N \times N$ matrix is deduced from the larger matrix. Indeed, every N-point DCT-2 ($N \times N$ matrix) is decomposed into two matrices: one referred to as the even decomposition and the second one as the odd decomposition. The even decomposition is actually the N/2-point DCT-2 ($N/2 \times N/2$ matrix). Figure 4.2 shows an example of the butterfly decomposition for 4-point DCT-2.

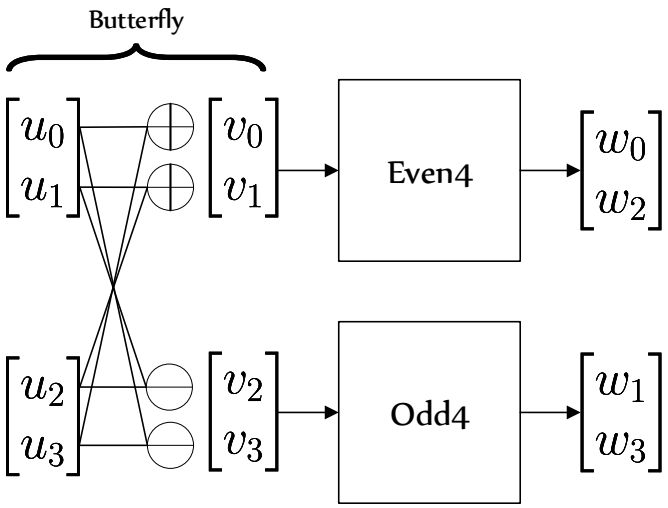


Fig. 4.2 Illustration of the butterfly decomposition for 4-point forward DCT-2.

Let $u = [u_0, u_1, u_2, u_3]^t$ be the input and be $w = [w_0, w_1, w_2, w_3]^t$ the output of the 4-point 1D forward transform, which is given by the following equation:

$$w = D_4\, u \tag{4.6}$$

Even-odd decomposition is given by Eq (4.7), (4.8), (4.9):
The addition and subtraction process (the butterfly) is shown by Eq (4.7).

$$\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} u_0 + u_3 \\ u_1 + u2 \\ u_2 - u_1 \\ u_3 - u_0 \end{bmatrix} \tag{4.7}$$

The even and odd part are shown in Eq (4.8) and (4.9), respectively.

$$\begin{bmatrix} w_0 \\ w_2 \end{bmatrix} = \begin{bmatrix} ev_0 & ev_1 \\ ev_2 & ev_3 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} \tag{4.8}$$

$$\begin{bmatrix} w_1 \\ w_3 \end{bmatrix} = \begin{bmatrix} odd_0 & odd_1 \\ odd_2 & odd_3 \end{bmatrix} \cdot \begin{bmatrix} v_2 \\ v_3 \end{bmatrix} \tag{4.9}$$

- **DCT-8/DST-7:** Same as the DCT-2, the transform core of DST-7/DCT-8 is scaled by a pre-defined factor and rounded to the nearest integer, or further tuned by an offset.

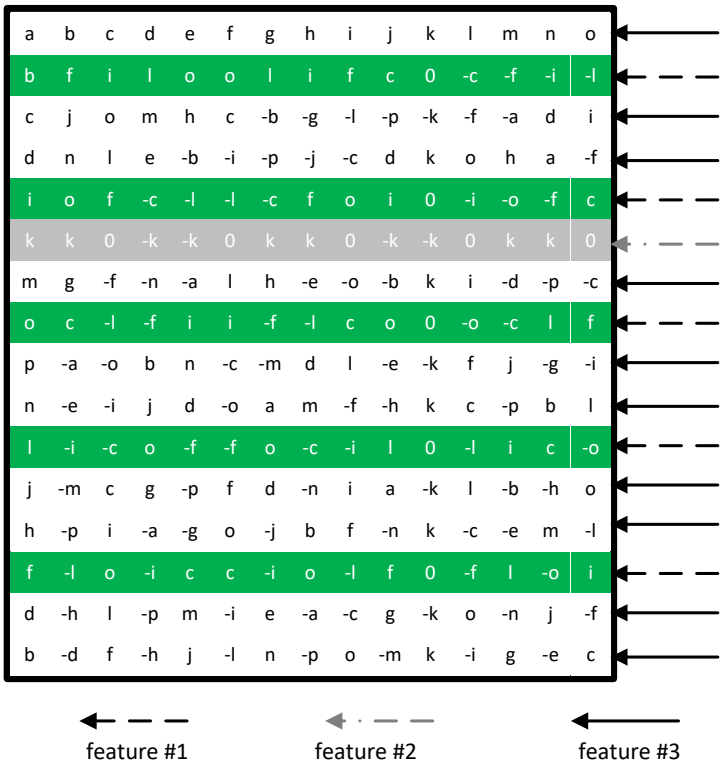


Fig. 4.3 Illustration of three features in the 16×16 DST-7 transform matrix of VVC [2].

The adjustment of DST-7/DCT-8 kernels are performed to ensure the three notable features associated with DST-7/DCT-8, as illustrated in Figure 4.3, including feature 1) repetitive segments {b, f , i, l, o} in some bases (interrupted black arrows), 2) unique coefficient value in one basis (interrupted gray arrow) and 3) mathematical relationship among coefficients with fixed pattern in some bases (black arrows), as pointed out in [2] and shown by the following equation:

$$l = a + j,\ m = b + i,\ n = c + h,\ o = d + g,\ p = e + f \tag{4.10}$$

Another important feature is that the matrices of different sizes of DST-7 and DCT-8 shares the same coefficient but with a certain twist. Eq (4.11) computes the DCT-8 C_8 from the DST-7 S_7 using pre-processing Λ and post-processing Γ matrices.

$$C_8 = \Lambda \cdot S_7 \cdot \Gamma, \tag{4.11}$$

where Λ and Γ are permutation and sign changes matrices computed by Eq (4.12) and (4.13), respectively.

$$\Lambda_{i,j} = \begin{cases} 1, & if\ j = N - 1 - i, \\ 0, & \text{otherwise}, \end{cases} \tag{4.12}$$

$$\Gamma_{i,j} = \begin{cases} (-1)^i, & if\ j = i, \\ 0, & \text{otherwise}, \end{cases} \tag{4.13}$$

with $i, j \in \{0, 1, \ldots, N-1\}$ and $N \in \{4, 8, 16, 32\}$.

Therefore, the DCT-8 is computed only by inverting the input order using a pre-processing stage and assigning the appropriate outputs signs using a post-processing stage.

In VVC, the MTS selects, for Luma blocks of size lower than 64, a set of transforms that minimizes the rate distortion cost an shown in Figure 4.4. Five different combinations of transform types are supported, including the conventional (DCT-2, DCT-2) and four new MTS mode combinations, i.e., (DST-7, DST-7), (DST-7, DCT-8), (DCT-8, DST-7) and (DCT-8, DCT-8). The explicit combination between DCT-2 and DST-7 (or DCT-8) with extra signalling overhead is not supported due to the limited coding gain and increased complexity for introducing extra encoder search and additional transform combinations. DST-7 and DCT-8 can be applied to the luma blocks. For chroma, the potential benefits of DST-7/DCT-8 have been studied during the development of VVC. However, since chroma components typically present smooth textures, where DCT-2 is sufficient, the coding gain versus complexity trade-off is less beneficial. Two variants of MTS are defined for VVC, called explicit and implicit MTS. The explicit MTS can be applied to both intra and inter coded blocks, while the implicit MTS can be only used for intra coded blocks. In explicit MTS, used by default in the reference software, the *tu_mts_idx* syntax element signals the selected horizontal and vertical transforms as specified in Table 4.1. In implicit MTS, the transform type are selected based on coded information that is known to both the encoder and decoder, and transform type signalling is not needed. The main benefit of implicit MTS is that it provides a significant coding gain as compared to explicit MTS, it provides significant coding gain over DCT-2 without any encoder search. This feature is appealing for simple encoder designs that cannot accommodate a complex rate-distortion search. To reduce the computational cost of large block–size transforms, the effective height M' and

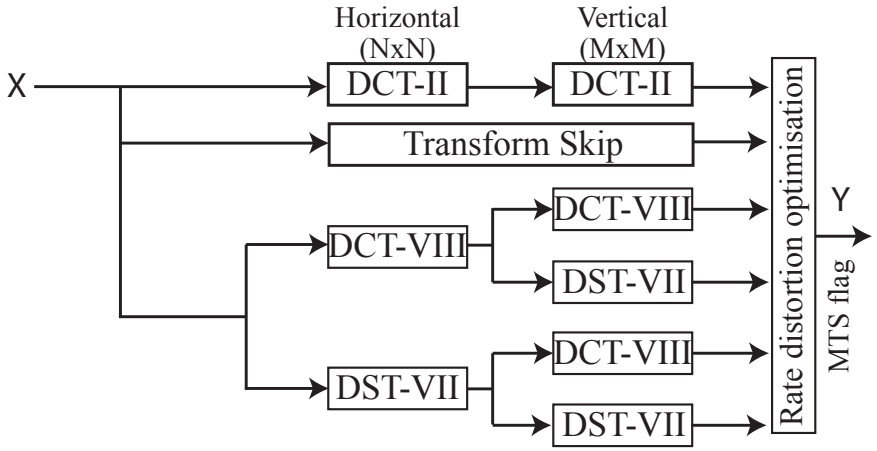


Fig. 4.4 The concept of 2D separable transforms selection in VVC. X is the input block of residuals, Y is the output transformed block and MTS flag is the index of the selected set of transforms. The DST-7 and DCT-8 transforms are used only for Luma samples of block size lower than 64.

width N' of the coding block (CB) are reduced depending of the CB size and transform type.

$$N' = \begin{cases} min(N,16) & trTypeHor > 0, \\ min(N,32) & \text{otherwise.} \end{cases} \tag{4.14}$$

$$M' = \begin{cases} min(M,16) & trTypeVer > 0, \\ min(M,32) & \text{otherwise.} \end{cases} \tag{4.15}$$

Table 4.1 Primary Transform signaling in VVC

tu_mts_idx	Transform Direction	
	Horizontal Transform	Vertical Transform
0	DCT-2	DCT-2
1	DST-7	DST-7
2	DCT-8	DST-7
3	DST-7	DCT-8
4	DCT-8	DCT-8

In (4.14) and (4.15), M' and N' are the effective width and height sizes, the terms $trTypeHor$ and $trTypeVer$ are respectively the types of vertical and horizontal transforms (0: DCT-2, 1: DCT-8 and 2: DST-7), and the $min(a,b)$ function returns the minimum between a and b. The sample value beyond the limits of the effective N and M are considered to be zero, thus reducing the computational cost of the 64-size DCT-2 and 32-size DCT-8/DST-7 transforms. This concept is called zeroing in the VVC standard. Figure 4.5 shows the possible zeroing scenarios for DCT-2 and DCT-8/DST-7 transforms.

4.3 Background of the separable transform (LFNST)

The LFNST [50, 51] has been adopted in the VTM version 5. The LFNST relies on matrix multiplication applied between the forward primary transform and the quantisation at the encoder side:

$$\vec{Z} = T \cdot \vec{Y}, \tag{4.16}$$

where the vector $\vec{Y}$ includes the coefficients of the residual block rearranged in a vector and the matrix T contains the coefficients transform kernel. The LFNST is enabled only when DCT-2 is used as a primary transform. The inverse LFNST is expressed in (4.17).

$$\vec{\tilde{Y}} = T^T \cdot \vec{\tilde{Z}}. \tag{4.17}$$

LFNST, which is known as reduced secondary transform, is applied between forward primary transform and quantization (at encoder) and between de-quantization and inverse primary transform (at decoder side) as shown in Figure 4.6. Different from the primary transform, the secondary transform

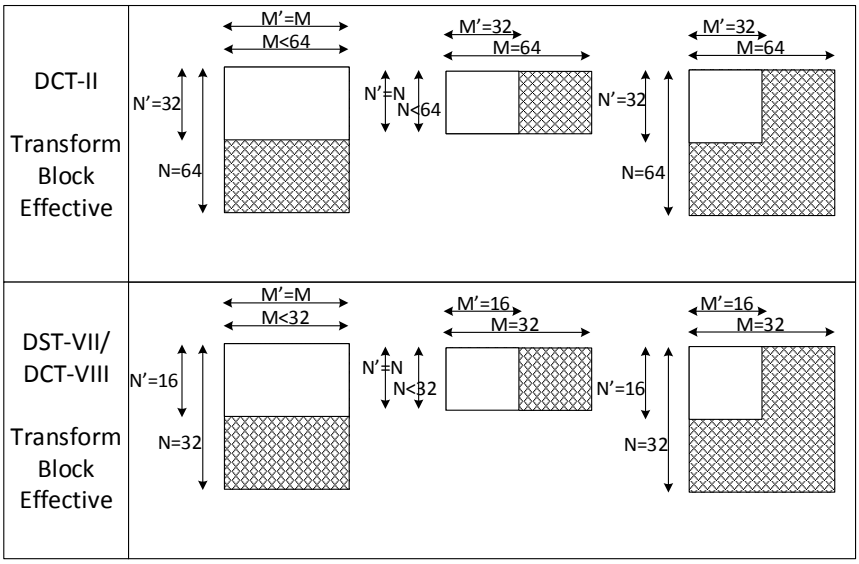


Fig. 4.5 Zeroing for large block sizes, M' and N' are the effective width and height sizes. The block area set to zero is illustrated in gray color.

operates only on the low-frequency transform coefficients. It is applied for all TU sizes from 4×4 to 64×64 including symmetric and asymmetric blocks. At the encoder side, it only process the top-left 8×8 corner of the TB, while at the decoder side, it process the top-left 4×4 corner. Four sets of two LFNST kernels of sizes 16×16 and 48×16 are applied on the top left 8×8 corner for the forward LFNST and on the top left 4×4 corner of the inverse LFNST, regardless of the input block size. Figure 4.7, shows in more details the input output processed block sizes for the forward and inverse LFNST. For instance, for the forward LFNST and for blocks of size 8×8, the size of the processed input equals 48 and the size of transformed output equals 8.

To apply the LFNST on a block of size 4×4, the 4×4 input block X, Eq (4.18), is first represented as a vector $\vec{X}$, as shown in Eq (4.19). The vectorization of X actually differs at the encoder and decoder. In Eq (4.19) the case for the decoder where X is raster scanned is considered, while, at the encoder the input vector is zig-zag scanned before it's feed to the forward LFNST. The vector X is then processed by the forward or inverse LFNST as presented previously in Eq (4.16) and (4.17).

$$X = \begin{bmatrix} X_{00} & X_{01} & X_{02} & X_{03} \\ X_{10} & X_{11} & X_{12} & X_{13} \\ X_{20} & X_{21} & X_{22} & X_{23} \\ X_{30} & X_{31} & X_{32} & X_{33} \end{bmatrix} \tag{4.18}$$

$$\vec{X} = [X_{00}, X_{01}, X_{02}, X_{03}, X_{10}, X_{11}, X_{12}, X_{13}, X_{20}, X_{21}, X_{22}, X_{23}, X_{30}, X_{31}, X_{32}, X_{33}] \tag{4.19}$$

There is in LFNST a total of 4 transform sets, with 2 non-separable transform matrices (kernels) for

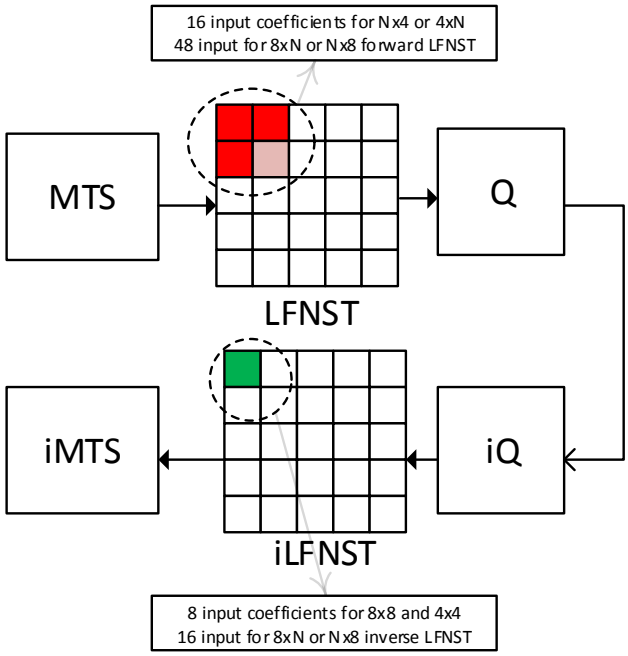


Fig. 4.6 Low-Frequency Non-Separable Transform (LFNST) process

each transform set. The mapping from the intra prediction mode (described in section 3.4.2) to the transform set is pre-defined as shown in Table 4.2. If one of three CCLM modes (INTRA_LT_CCLM, INTRA_T_CCLM or INTRA_L_CCLM) is used for the current block ($81 <= IPM <= 83$), transform set index 0 is selected for the current chroma block. For each transform set, the selected non-separable transform candidate is further specified by the explicitly signalled LFNST index. The index is signalled in a bit-stream once per Intra CU after transform coefficients.

Table 4.2 Intra Prediction Mode (IPM) based secondary transform signaling in VVC

Intra Prediction Mode	Transform set index
$IPM < 0$	1
$0 \leq IPM \leq 1$	0
$2 \leq IPM \leq 12$	1
$13 \leq IPM \leq 23$	2
$24 \leq IPM \leq 44$	3
$45 \leq IPM \leq 55$	2
$56 \leq IPM \leq 80$	1
$81 \leq IPM \leq 83$	0

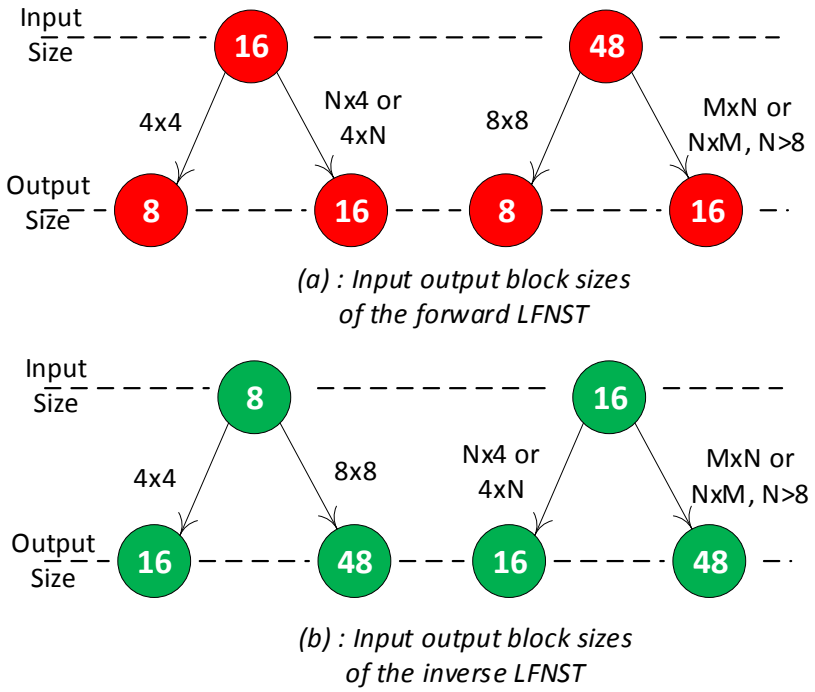


Fig. 4.7 LFNST input/output block sizes at the encoder and decoder sides

4.4 State-of-the-art on hardware implementations of separable transforms

In this section, a brief description and analysis of existing hardware implementations of separable transforms proposed for HEVC and VVC is provided.

4.4.1 Hardware Implementations of DCT-2

DCT-2 transform has been widely used by previous video coding standards such as AVC and HEVC. Therefore, its hardware implementation has been well studied and optimized for different architectures. Shen *et al.* [52] proposed a unified Very Large Scale Integration (VLSI) architecture for 4, 8, 16 and 32 point Inverse Integer Core Transforms (IICT). This latter relies on shared regular multipliers architecture that takes advantage of the recursion feature for large size blocks of 16- and 32-points IICTs. The proposed solution relies on a Static Random Access Memory (SRAM) to transpose the intermediate 1-D transform results which, in result, reduces logic resources. Zhu et *al.* [53] proposed a unified and fully pipelined 2D architecture for DCT-2, IDCT-2 and Hadamard transforms. This architecture also benefits from hardware resources sharing and a SRAM module to transpose the result of the intermediate 1D transform. Meher *et al.* [54] proposed a reusable architecture for DCT-2 supporting different transform sizes using constant multiplications instead of regular ones. This architecture has a fixed throughput of 32 coefficients per cycle regardless the transform size and it can be pruned to reduce the implementation complexity of both full-parallel and folded 2-D DCT-2.

However, this approximation leads to a certain degradation on the coding performance varying from 0.8% to 1% of Bjøntegaard Delta Rate (BD-BR) losses when both inverse DCT-2 and DCT-2 are pruned. Chen *et al.* [55] proposed a 2D hardware design for the HEVC DCT-2 transform targeting Field Programmable Gate Array (FPGA) platform. The presented re-configurable architecture supports up to 32×32 transform block sizes. To reduce logic utilization, this architecture benefits from several hardware resources, such as Digital Signal Processing (DSP) blocks, multipliers and memory blocks. The proposed architecture has been synthesized targeting different FPGA platforms showing that the design is able to encode 4Kp30 video at a reduced hardware cost. Ahmed *et al.* [56] proposed a dynamic N-point inverse DCT-2 hardware implementation supporting all HEVC transform block sizes. The proposed architecture is partially folded in order to save area and speed up the design. This architecture reached an operating frequency of 150 MHz and supports real time processing of 1080p30 video.

4.4.2 Hardware Implementation of the MTS block

Several works [57–63] have recently investigated a hardware implementation of the earlier version of the MTS that includes several transform types. Mert *et al.* [57] proposed a 2D hardware implementation including all five transform types for 4-point and 8-point sizes using additions and shifts instead of regular multiplications. This solution supports a 2D hardware implementation of the five transform types. However, transform sizes larger than 8 are not supported, which are more complex and would require more resources. A pipelined 1D hardware implementation for all block sizes from 4×4 to 32×32 was proposed by Garrido *et al.* [58]. However, this solution only considers 1D transform, while including the 2D transform would normally be more complex. Moreover, this design does not consider asymmetric block size combinations. This latter design has been extended by Garrido *et al.* [59] to support 2D transform using Dual port Random-Access Memory (RAM) as a transpose memory to store the 1D intermediate results. Authors proposed a pipelined 2D design placing two separate 1D processors in parallel for horizontal and vertical transforms. A multiplierless implementation of the MTS 4-point transform module was proposed by Kammoun *et al.* [60]. This solution has been extended to 2D hardware implementation of all block sizes (including asymmetric ones), by using the Intellectual Property (IP) Cores multipliers [64] to leverage the DSPs blocks of the *Arria 10* platform [61]. This solution supports all transform types and enables a 2D transform process within an pipelined architecture. However, this design requires a high logic resources compared to solution proposed by Mert *et al.* [57] and Garrido *et al.* [58]. The approximation of the DST-7 based on an inverse DCT-2 transform and low complexity adjustment stage was investigated in [65]. This solution supports a unified architecture for both forward and inverse MTS with moderate hardware resources. However, the approximation introduces a slight bitrate loss while the architecture is not compliant with a VVC decoder. Fan *et al.* [63] proposed a pipelined 2D transform architecture for DCT-8 and DST-7 relying on the N-Dimensional Reduced Adder Graph (RAG-N) algorithm. The use of this algorithm enables an efficient use of adders and shifts to replace regular multipliers. However, this work does not include the support of DCT-2 which requires high logic resources especially for large size block (ie. 64×64). The only work found in the literature that supports all MTS types and sizes in 2-D was proposed in [62]. This work is an extension of the architectures proposed by Garrido *et*

al. [58,59]. This solution supports all the MTS block sizes up to 64×64 including asymmetric blocks. However, the performance of the proposed design is considered to be low especially for a nominal scenario (worst case). The proposed architecture enables decoding UHD videos at 10 Frame Per Second (FPS). Moreover, considering the worst case scenario, this solution is only able to decode HD videos at 32 FPS. These architectures are summarized and compared with our proposed solution in table 5.5 of Chapter 5.

4.5 Conclusion

The VVC transform module raises many challenges to the hardware implementation of the VVC decoder. The introduced DCT-2 (p-64), DST-7, DCT-8 kernels with the 8 LFNST kernels will require high memory usage to store these coefficients. Moreover, these new transforms are more complex and require a higher number of multiplications. Finally, the LFNST stage was not previously studied in literature and introduces an additional delay required to perform the secondary transform. Therefore, the hardware transform module should be carefully designed leveraging all optimisations to reach the required latency and throughput while minimising the hardware area.

Chapter 5

HW Design of MTS for 4K VVC Decoder

5.1 Introduction

Multiple Transform Selection (MTS) is one of the new concepts introduced in Versatile Video Coding (VVC). The final version of the MTS, integrated in the Joint Video Experts Team (JVET), consists of three transform kernels including Discrete Cosine Transform (DCT) types 2 and 8 , and Discrete Sine Transform (DST) types 7. The basis functions of these transforms are expressed by Equations (4.3), (4.4) and (4.5) in Chapter 4 for DCT-2 C_2, DST-7 S_7 and DCT-8 C_8, respectively. The real-time video decoder should have the throughput sufficiently high to support required resolutions. In this work, the aim is to design a real time VVC video decoding chip that supports 4kp60 video resolution with a target frequency of 450Mhz. The Application-Specific Integrated Circuit (ASIC) chip is expected to decode the target resolution over two cores. As a result, each core is expected to process 4Kp30 resolution in real time. In practice, the required throughput for a specific tool of the VVC decoder depends on the video resolution specified in time and pixel domains, which are measured with respect to the Frame Per Second (FPS) and the pixel area, respectively. Sub-sampling of chroma components can additionally affect the throughput performance. Since the MTS module processes residuals in $N \times N$ pixel units, it is convenient to use their number per second to specify and measure the throughput.

Table 5.1 shows the detailed performance for each block size for the inverse MTS. It highlights the cycle budget available for each input block and the required number of processed residuals per cycle with respect to the target frequency. To show the impact of the system frequency on the overall performance and the throughput we give two frequency examples (100Mhz and 450Mhz). However, for our decoding chip, the target frequency is 450Mhz. As we can notice from Table 5.1, as the system frequency increases the clock budget per unit increases. Therefore, the available cycles per unit increase and the total number of residuals per cycle to process decreases. This impacts the size of the ASIC chip, as we increase the cycle budget per unit we are allowing for more space for design pipelining and for resource sharing, which results in decreasing the area of the chip. Therefore, the chip size increases when the frequency decreases.

In practice, the performance should have a computation margin to compensate for wait states introduced by control and communication operations, e.g., initialization or access to external memories. Therefore, numbers of clock cycles given in Table 5.1 should be slightly lower. For example, it is use-

Table 5.1 2-D iMTS throughput performance bookmarking per block size for 4Kp30:420 resolution

Unit size	Throughput (unit/s)	DCT2				DST7/DCT8			
–	–	100 Mhz		450 Mhz		100 Mhz		450 Mhz	
–	–	cycle budget (cycles)	sample/cycle	cycle budget (cycles)	sample/cycle	cycle budget (cycles)	sample/cycle	cycle budget (cycles)	sample/cycle
64×64	91125	1097	1	4938	0.25	✗	✗	✗	✗
64×32,32×64	182250	548	2	2469	0.5	✗	✗	✗	✗
64×16,16×64	364500	274	2	1234	0.5	✗	✗	✗	✗
64×8,8×64	729000	137	2	617	0.5	✗	✗	✗	✗
64×4,4×64	1458000	68	2	308	0.5	✗	✗	✗	✗
32×32	364500	274	4	1234	1	274	1	1234	0.25
32×16,16×32	729000	137	4	617	1	137	2	617	0.5
32×8,8×32	1458000	68	4	308	1	68	2	308	0.5
32×4,4×32	2916000	34	4	154	1	34	2	154	0.5
16×16	1458000	68	4	308	1	68	4	308	1
16×8,8×16	2916000	34	4	154	1	34	4	154	1
16×4,4×16	5832000	17	4	77	1	17	4	77	1
8×8	5832000	17	4	77	1	17	4	77	1
8×4,4×8	11664000	8	4	38	1	8	4	38	1
4×4	23328000	4	4	19	1	4	4	19	1

ful to assume that the number of available cycles for 16×16 units for frequency 450 Mhz is equal to 300 rather than 308. For large block sizes, we can notice that the number of required residuals per clock cycle (sample/cycle) is four times less than the case for smaller block sizes. This is due to the zeroing for large block sizes used for the VVC transform. In fact, high frequency transform coefficients are zeroed out for the transform blocks of sizes equal to 64 for DCT-2 and 32 for transform types DST-7/DCT-8. Therefore, only lower-frequency coefficients are retained. As a result, three quarters of the 64×64 block for DCT-2 are actually zeros and are not transformed at the input of the inverse MTS. The same goes for 32×32 blocks for the DST-7/DCT-8 transform types, where only one quarter of the input block is processed. Similarly for asymmetrical blocks, only half of the information is transformed for $64 \times N$ and $N \times 64$ $(N < 64)$ block sizes when applying DCT-2 and for $32 \times N$ and $N \times 32$ $(N < 32)$ when applying DST-7/DCT-8. However, to support all transform types and sizes, the performance of the 2-D MTS is measured on the worst case scenario. In this case, the 2-D MTS module should sustain a fixed throughput of 1 sample/cycle. Furthermore, to facilitate chaining between different block sizes, the latter should sustain a unified latency for all block sizes and transform types. This latency corresponds to the largest clock cycle budget equals to 4938 cycles, which is measured for 64×64 block size, and because only one quarter of block is processed the real budget is 1234 clock cycles.

To save hardware resources, the 2-D MTS is implemented in folded architecture as shown in Figure 5.1. It is composed of the two sub-modules: the 1-D MTS that computes the 1D transform for one row/column vector and the transposition buffer. All results of the first 1D transform must be written to the buffer before the second transform direction starts. In the first step, the sub-module takes the input residuals and writes the result in the transpose buffer. In the second step, the sub-module receives the input from the transpose memory and writes the 2D-transformed block to the output memory. As a result, the throughput of the 1D MTS sub-module is twice of that offered by the entire folded architecture. Therefore, to sustain a fixed throughput of 1 sample/cycle for the 2D MTS, the 1D MTS should sustain a throughput of 2 sample/cycle . The complexity of the transforms in H.266/VVC is much higher since larger Transform Unit (TU) sizes are supported (up to 64×64 for

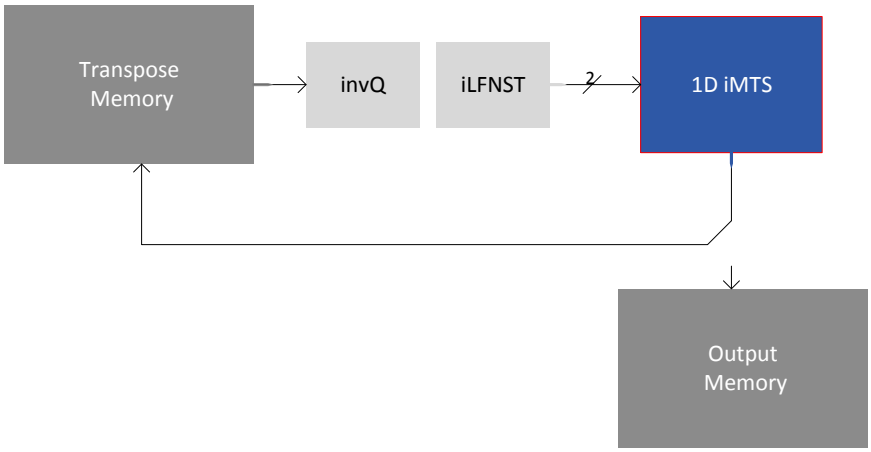


Fig. 5.1 2D VVC transform in a folded architecture

DCT-2 and 32×32 for DCT-8/DST-7) compared to the one for H.265/HEVC. This fact increases the size of the transpose buffer which should contain a 64×64 Coding Tree Unit (CTU). Thus, since one decoding core is expected to process 4Kp30 resolution for 4:2:0 Chroma sub-sampling, the memory should support two Chroma blocks of size 32×32 and one Luma block of size 64×64.

This chapter tackles the problem of hardware implementation of the inverse 1D VVC MTS targeting 4Kp30 resolution and 450Mhz frequency. The 1-D MTS should sustain a fixed system latency with a fixed throughput of 2 sample/cycle. Almost, every state of the art study for the early transform Hardware (HW) design considers mostly a multiplier-less approach (a.k.a Constant Multipliers (CM) instead of using Regular Multiplier (RM). In this work, we present a study for two hardware architectures for the VVC MTS. The first one implements the transform using a CM based architecture, while the second one uses a RM based architecture. The MTS processor interfaces for both architectures is summarized in Table 5.2. A positive pulse in $input\_en$ launches the transform process, with the size, type and transform direction are defined in the tr_size, tr_type and tr_dir input signals, respectively. Following the $input\_en$ the $data\_in$ signal will carry four coefficients over two clock cycles (2 coefficients each cycle). After the computation, outputs can be read from the $data\_out\_inter$ or $data\_out\_final$ signals, depending on the transform direction, at a speed of two coefficients each clock cycle. The N_{bi} and N_{bo} represent the output bit size for the first and second direction, respectively. For both implementations that integrate data loading and pipeline stages, the design starts generating the result of 1-D row/columns DCT/DST after a fixed system latency. It then continues generating the outputs every clock cycle without any stalls. Through this study we show that the use of RM based HW design is more efficient in terms of surface area for the VVC MTS, unlike previous study for previous standards. This is due to the fact that the VVC MTS uses a wider range of transform types and larger transform block sizes.

The following sections are organized as follows. The proposed hardware design for both CM and RM based architecture are described in Section 5.2 and Section 5.3, respectively. Section 5.4 presents the the experimental setup and the synthesis results of 1D CM and RM implementations. Finally, Section

Table 5.2 MTS module interface

Signal	I/O	#BITS	Description
clk	I	1	System Clock
rst_n	I	1	System reset, active low
input_en	I	1	Activation pulse to start
tr_type	I	2	Transform type: 0, DCT-2; 1, DCT-8; 2,DST-7
tr_size	I	3	Size: 0:4-pnt; 1:8-pnt; 2:16-pnt; 3:32-pnt;4:64-pnt
tr_dir	I	1	Direction: 0:Horizontal; 1: Vertical
data_in	I	$4\times N_{bi}$	Input data
out_en	O	1	Activation pulse to indicate end of N-point
data_out_inter	O	$4\times N_{bi}$	Intermediate output data
data_out_fin	O	$4\times N_{bo}$	Final result

5.5 concludes the chapter.

5.2 Constant Multiplier based MTS architecture

Diagram 5.2 depicts the top level 1D MTS module CM-based architecture: As we can notice from Figure 5.2, the top level design is composed of five sub-modules: 64-point iDCT-2, 4-point iDCT-8/iDST-7, 8-point iDCT-8/iDST-7, 16-point iDCT-8/iDST-7 and 32-point iDCT-8/iDST-7. Using its butterfly feature, every N-point iDCT-2 is computed using only the 64-point iDCT-2. This is because smaller transform blocks are embedded within larger ones. On the other hand, the DCT-8 and DST-7 transform types do not have the same feature as the DCT-2. Therefore, each N-point iDCT-8/iDST-7 is implemented separately.

5.2.1 iDCT-2 unified hardware architecture

As explained in Chapter 4, thanks to the butterfly propriety of the DCT-2, its operation can be computed in a recursive manner. An N point 1D transform can be performed by applying two N/2-point 1D transforms with an additional post-processing stage. Every N-point transformation is decomposed into two N/2-point transforms, one referred to as the even decomposition and the second as the odd one. In fact, the 1-D iDCT-2 can be represented as shown in equations (5.1) and (5.2):

$$Y_n = C_n^T \cdot X_n, \tag{5.1}$$

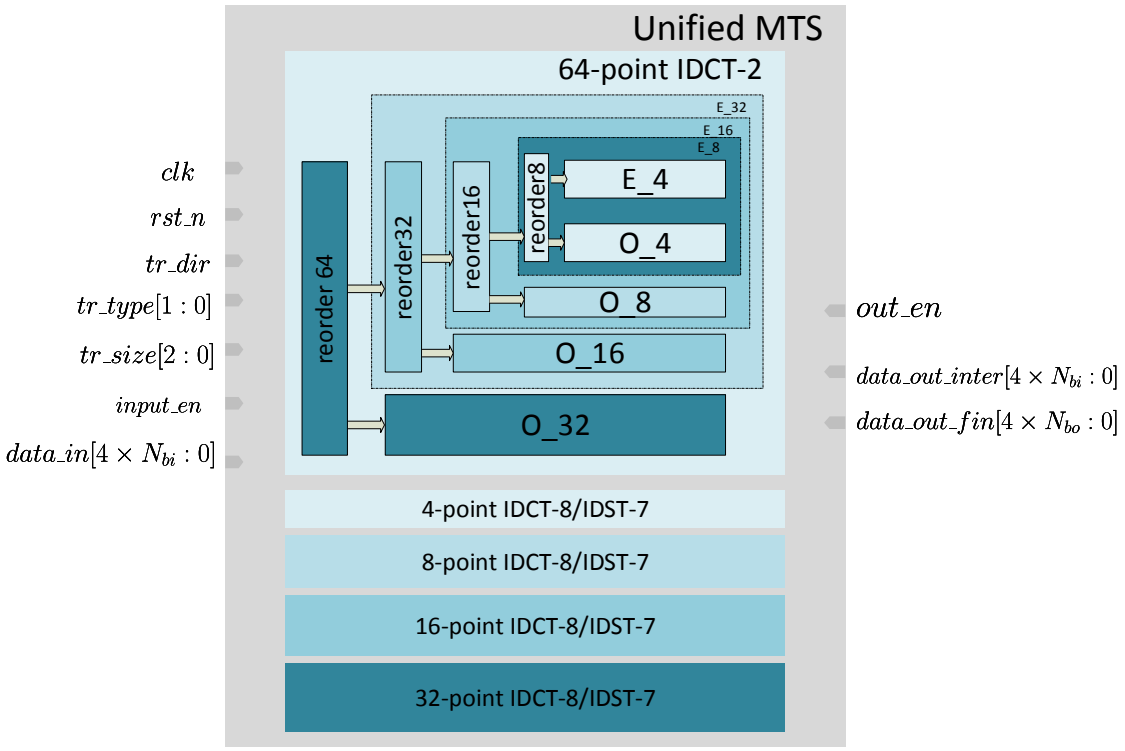


Fig. 5.2 Inverse CM-based MTS top level block diagram

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} C_{0,0} & C_{0,1} & \dots & C_{0,N-1} \\ C_{1,0} & C_{1,1} & \dots & C_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{N-1,0} & C_{00} & \dots & C_{N-1,N-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{5.2}$$

where Y_n represents the output vector and X_n is the input vector of size N. C_n^T represents the inverse transform matrix of size $N \times N$. The basic feature of the transformation matrix is that the columns' elements of the even rows are mirrored (i.e. for N-point iDCT-2, column element with index i in an even row has the same values of the column element with index $N-1-i$ in the same row) and the columns' elements of the odd rows are negatively mirrored in the same manner. This feature is very helpful in reducing the calculations by adding or subtracting the outputs that have a common multiplier. Using this feature, Eq (5.1) can be rewritten to what is shown in Eq (5.3) and (5.4) where $Y_{low} = Y_{i=0\dots\frac{N}{2}-1}$ and $Y_{high} = Y_{i=\frac{N}{2}\dots N-1}$.

$$Y_{low} = C_{n/2}^T \cdot X_{even} + O_{n/2}^T \cdot X_{odd}, \tag{5.3}$$

$$Y_{high} = C_{n/2}^T \cdot X_{even} - O_{n/2}^T \cdot X_{odd}, \tag{5.4}$$

where the elements values of the matrices $C^T_{n/2}$ and $O^T_{n/2}$ are defined as:

$$C^T_{n/2}(i,j) = C^T_n(i,2j), \;\; O^T_{n/2}(i,j) = C^T_n(i,2j+1) \quad \text{,with } 0 \leq i,j \leq N/2\check{}1 \tag{5.5}$$

Therefore, the N-point inverse DCT-2 is composed of two main stages: the core computation (for the odd and even parts) and the reordering stage. Figure 5.3 illustrates the top level design of an N-point iDCT-2 module. Depending on the position of input elements, the even and odd sub-modules will be feed from the input vector X.

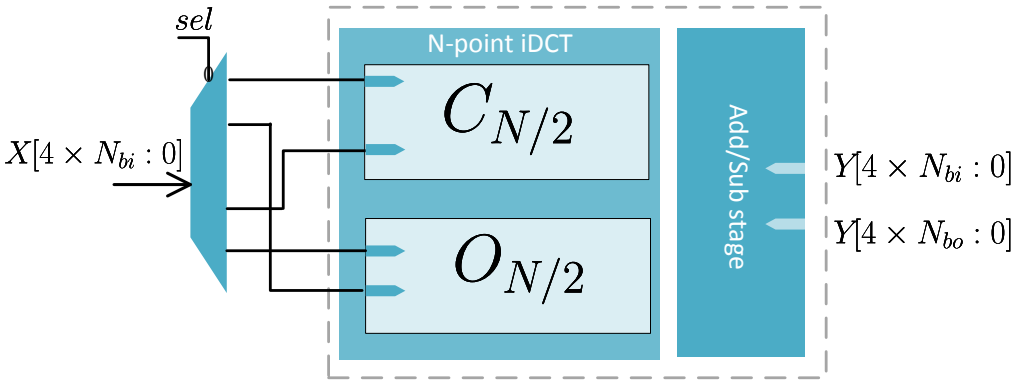


Fig. 5.3 N-point inverse DCT-2 top level design

The N-point iDCT computes the following Equations separately (derived from Eq (5.3)):

$$A = C^T_{n/2} \cdot X_{even} \quad B = O^T_{n/2} \cdot X_{odd}, \tag{5.6}$$

In this CM-based architecture, the multiplications using the odd or the even sub-modules are implemented without the use of multipliers. Every coefficient of the transform matrix is computed through the use of addition and shift operations only. Using the Butterfly decomposition, the 64-point iDCT-2 matrix is represented in Figure 5.4. The 64×64 can be decomposed in a total of five sub-matrices. Figure 5.4 shows the example of 16×16 butterfly decomposed DCT-2 matrix. Each sub-matrix of the 64×64 has a set of unique operators that are replicated with a positive or negative sign and they are as follows:

- For odd 64×64 (O32) the operators are $[91,90,90,90,88,87,86,84,83,81,79,77,73,71,69,65,62,59,56,52,48,44,41,37,33,28,24,20,15,11,7,2]$
- For odd 32×32 (O16): the operators are $[4,13,22,31,38,46,54,61,67,73,78,82,85,88,90,90]$
- For odd 16×16 (O8): the operators are $[9,25,43,57,70,80,87,90]$
- For odd 8×8 (O4): the operators are $[18,50,75,89]$
- For odd 4×4 (O2): the operators are $[36,83]$
- For even 4×4 (E2): the operators are $[64]$

64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	36	83	0	0	0	0	0	0	0	0	0	0	0	0
0	0	-83	36	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	18	50	75	89	0	0	0	0	0	0	0	0
0	0	0	0	-50	-89	-18	75	0	0	0	0	0	0	0	0
0	0	0	0	75	18	-89	50	0	0	0	0	0	0	0	0
0	0	0	0	-89	75	-50	18	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	9	25	43	57	70	80	87	90
0	0	0	0	0	0	0	0	-25	-70	-90	-80	-43	9	57	87
0	0	0	0	0	0	0	0	43	90	57	-25	-87	-70	9	80
0	0	0	0	0	0	0	0	-57	-80	25	90	9	-87	-43	70
0	0	0	0	0	0	0	0	70	43	-87	-9	90	-25	-80	57
0	0	0	0	0	0	0	0	-80	9	70	-87	25	57	-90	43
0	0	0	0	0	0	0	0	87	-57	9	43	-80	90	-70	25
0	0	0	0	0	0	0	0	-90	87	-80	70	-57	43	-25	9

Fig. 5.4 16-point iDCT2 butterfly decomposition

For the multiplication stage, the multiplications are replaced with adders and shifts. For instance, in order to transform 8-point iDCT-2, sub-matrix O4, O2 and E2 are used. Figure 5.5 shows the diagram of 4-point iDCT-2 according to Eq (5.7) and (5.8) that shows how the multiplication operations are replaced with adders and shifts. It shows the data path for the input vector and the multiplication operators.

$$\begin{bmatrix} y_0 \\ y_2 \end{bmatrix} = \begin{bmatrix} 64 & 0 \\ 0 & 64 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, \begin{cases} \tilde{y_0} = (x_0 \ll 6) \\ \tilde{y_2} = (x_1 \ll 6) \end{cases} \tag{5.7}$$

Given that $36 = 2^5 + 2^2$ and $83 = 2^6 + 2^4 + 2^1 + 2^0$, the multiplication operations can be rewritten as shown in Eq (5.8).

$$\begin{bmatrix} y_1 \\ y_3 \end{bmatrix} = \begin{bmatrix} 36 & 83 \\ -83 & 36 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ x_3 \end{bmatrix}, \begin{cases} \tilde{y_1} = [(x_2 \ll 3 + x_2) \ll 2] + [(x_3 \ll 1 + x_3) + ((x_3 \ll 2 + x_3) \ll 4)] \\ \tilde{y_3} = [(x_3 \ll 3 + x_3) \ll 2] - [(x_2 \ll 1 + x_2) + ((x_2 \ll 2 + x_2) \ll 4)] \end{cases} \tag{5.8}$$

where $\tilde{y_i}$ represents the output vector of the multiplication stage and the input of the final add and subtract unit.

Then, at the next stage, the following operations are executed and they are depicted as shown in Figure 5.6:

$$Y_{i=0\ldots N/2-1} = A + B, \quad Y_{i=N/2\ldots N-1} = A - B \tag{5.9}$$

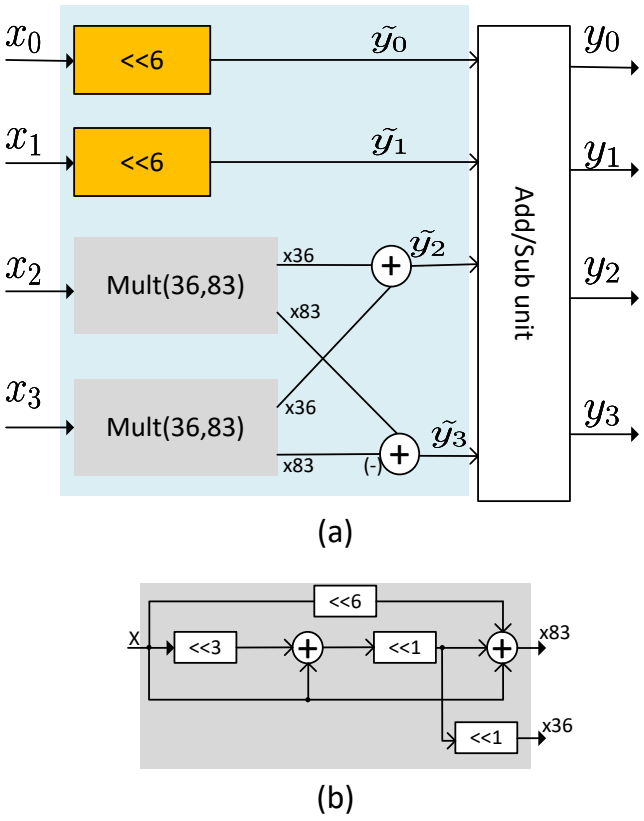


Fig. 5.5 Proposed design for the 4-point iDCT-2 using constant multipliers(a): 4-point iDCT-2 architecture (b): add and shift unit for the 36 and 83 operators

5.2.2 N-point iDST-7/iDCT-8

The CM-based architecture contains four separable modules for 4-point, 8-point, 16-point and 32-point iDCT-8/iDST-7. For the iDCT-8 and iDST-7 which do not have any specific decomposition, we propose one module for each block size including 4-point, 8-point, 16-point and 32-point. The relationship between iDCT-8 and iDST-7 enables to use only the iDST-7 kernel to compute both transformations by adding two stages for pre-processing and post-processing as shown in Figure 5.7. Eq (5.10) computes the iDCT-8 C_8^T from the iDST7 S_7^T using pre-processing Λ and post-processing Γ matrices.

$$C_8^T = \Lambda \cdot S_7^T \cdot \Gamma, \tag{5.10}$$

where Λ and Γ are permutation and sign changes matrices computed by Eq (5.11) and (5.12), respectively.

$$\Lambda_{i,j} = \begin{cases} 1, & if\ j = N-1-i, \\ 0, & \text{otherwise}, \end{cases} \tag{5.11}$$

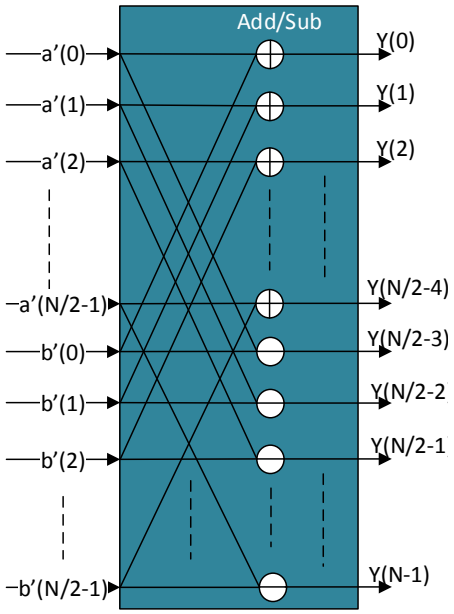


Fig. 5.6 N-point Add/Sub unit

$$\Gamma_{i,j} = \begin{cases} (-1)^i, & \text{if } j = i, \\ 0, & \text{otherwise}, \end{cases} \tag{5.12}$$

with $i, j \in \{0, 1, \ldots, N-1\}$ and $N \in \{4, 8, 16, 32\}$.

Therefore, the iDCT-8 is computed only by inverting the input order using a pre-processing stage and assigning the appropriate outputs signs using a post-processing stage. The computation part using the core iDST-7 module is calculated using adders and shifters instead of RM similar to the iDCT-2 implementation depicted in Figure 5.5 for 4-point transform size example.

5.3 Regular Multiplier based MTS architecture

The RM-based architecture uses a number of regular multipliers in a shared design with all transform types and sizes. All transform sizes of 4/8/16/32-point iDCT-2/8, iDST-7 and 64-point iDCT-2 use a total of 32 shared multipliers. Thirty-two is the maximum number of multiplications needed to get a throughput of 2 sample/cycle. This number is bounded by the odd decomposition of the 64×64 iDCT-2 transform matrix and the 32×32 iDST-7/iDCT-8 transform matrices. For large block sizes and by taking advantage of the zeroing, processing one sample/cycle at the input to get a 2 sample/cycle at the output is beneficial. In the case of 64-point iDCT-2, the size of the non zero input vector is 32 and the output vector size is 64. For the 32-point iDCT-8/iDST-7, the size of the non zero input and output vectors are 16 and 32, respectively. In both cases, the output vector is twice the size of the input vector, and thanks to this, the input rate can be lowered to one sample per cycle.

Figure 5.8 shows the top level design of the RM-based MTS module which is composed of three

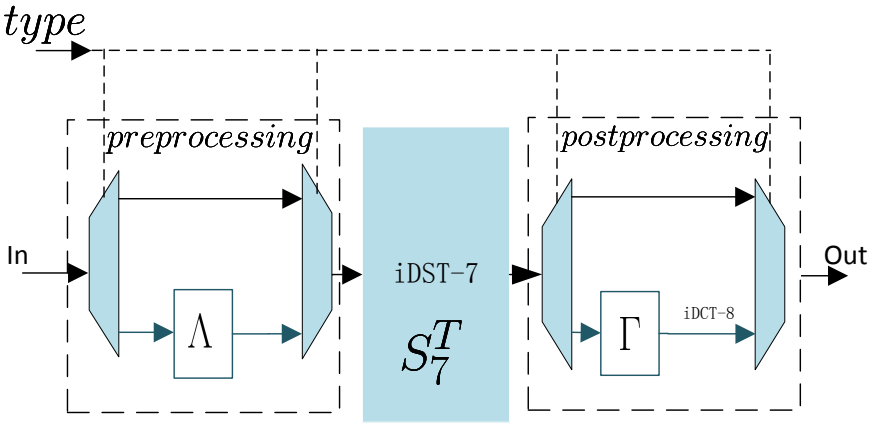


Fig. 5.7 N-point iDCT-8/iDST-7 hardware design using iDST-7 as kernel transformation, pre-processing and post-processing blocks are used when iDCT-8 type is active.

sub-modules: Read-Only Memory (ROM), Coeff-prepa, Multi-module.

5.3.1 Read Only Memory (ROM)

ROM is mainly used to store static data. For the MTS, the ROM is used to store the transform coefficients. These coefficients make a total of 1872 coefficients for all transform sizes and types. The ROM contains the 64-pt DCT-2, 32-pt, 16-pt, 8-pt and 4-pt DST-7. We take advantage of the butterfly decomposition to store the DCT-2 coefficients. The first butterfly decomposition produces two 32×32 matrices. The even matrix is further decomposed into two 16×16 matrices, which is further decomposed until it stops at 4×4 matrix of the 4-point DCT-2.

All MTS matrices are stored side by side. However, one exception is for 16×16 odd decomposition for the 32-point DCT-2 which is duplicated in the ROM due to the speed of the MTS module. These two copies are actually stored one on top of the other and sided by 32×16 matrix of the 64×64 odd decomposition. The other DST-7 transform matrices are stored side by side. Figure 5.9, shows the ROM structure and regions. As we can notice from the figure, there are nine regions in the memory and their content is organized as follows:

- R1: the 4×4 of 4-pt DCT2 and the 4×4 odd decomposition of the 8-pt DCT2 matrices.
- R2: the 16×16 DCT2 matrix.
- R3: a copy of the 16×16 DCT2 matrix.
- R4: the 16×16 odd decomposition of the 32-pt DCT2 matrix.
- R5: the 16×32 odd decomposition of the 64-pt DCT2 matrix.
- R6: the 4×4 DST7 matrix.

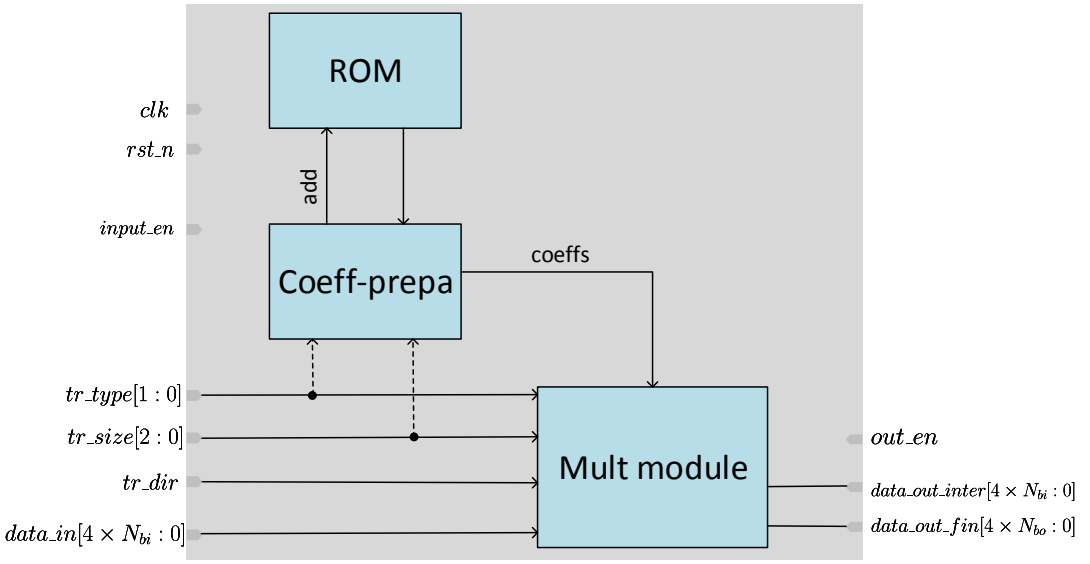


Fig. 5.8 Inverse RM-based MTS top level block diagram

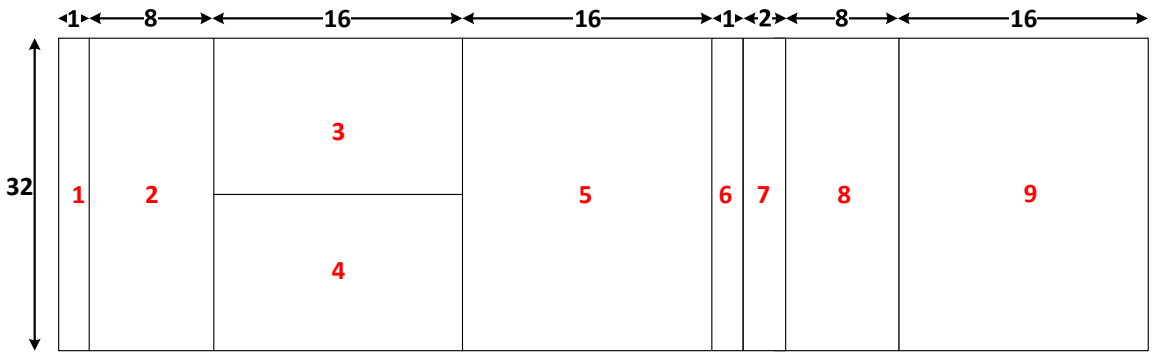


Fig. 5.9 Read Only Memory divided into regions

- R7: the 8×8 DST7 matrix.
- R8: the 16×16 DST7 matrix.
- R9: the 16×32 DST7 matrix.

The main reason behind this division is to efficiently handle the MTS rate, by keeping one memory access for every 2 input samples. As shown, the ROM height is 32 coefficients where each coefficient is represented by 8-bits. Therefore, the total memory size is 17408-bits which is 68 columns of 256-bit (32 × 8-bits). Finally, a 7-bit address bus is required to cover the entire memory space.

5.3.2 Coefficients preparation module

The coefficients preparation module is an in-between modules that selects the appropriate set of coefficients from the ROM, and delivers them to the multiplication module. Using the transform type and

the block size, the coefficient preparation module decides which ROM region is accessed using the appropriate address. Table 5.3 shows the mapping between the input state and the memory addresses and the data of accessed regions.

Table 5.3 Coefficient preparation module access and address definitions

Input		Output	
Tr type	block size	ROM address	data retrieved
0 (DCT-2)	0 (4)	0x0	4×4 DCT-2 matrix
0	1 (8)	0x0	4×4 even and odd matrices of 8-pt DCT-2
0	2 (16)	0x1 to 0x8	16×16 DCT-2 matrix
0	3 (32)	0x8 to 0x19	16×16 even and odd matrices of 32-pt DCT-2
0	4 (64)	0x8 to 0x29	16×16 even and odd matrices of 32-pt DCT-2 + 32×16 odd matrix of 64-pt DCT-2
1 or 2 (DST-7 or DCT-8)	0 (4)	0x2A	4×4 DST-7 matrix
1 or 2	1 (8)	0x2B to 0x2C	8×8 DST-7 matrix
1 or 2	2 (16)	0x2D to 0x34	16×16 DST-7 matrix
1 or 2	3 (32)	0x35 to 0x43	32×16 DST-7 matrix

5.3.3 Multiplication module

The multiplication module is the last stage of the RM-based architecture. It receives the input data along with the transform coefficients ROM the coeff_prepa module. In order to sustain a fixed throughput of 2 sample/cycle, the MTS mult_module uses a total of 32 RMs. The received coefficients are divided over two pools of 16 coefficients. Each of these pools is dedicated for one input residuals. Figure 5.10 shows the pools associated to the input vector at every clock cycle.

The green and blue colors in the figure indicate the two different pools for the two input samples. For every clock cycle, the two input samples are multiplied with coefficients belonging to the two different pools. Each sample is multiplied to all the coefficient of a pool. The maximum size of one pool is 16 coefficients. For the transform size that requires less than 16 coefficients, the remaining of the pool is filled up by zeros. Figure 5.10 shows the coefficients distribution over the two pools for the case of 4-point transform (regardless of the transform type) and for 8-point iDCT-2 when using the butterfly decomposition. Figure 5.11 shows the architecture of the hardware module using 32 RMs referred to m_i. X_0 and X_1 are the input samples. In case of large transform sizes, X_0 and X_1 interfaces will carry the same input sample and *sel* signal is disabled, otherwise they carry two different samples and *sel* is enabled. The input samples are then multiplied by the corresponding transform coefficients C_i $(i \in 1..N)$. C represents a line from the transform matrix spread out over two pools of 16 coefficients. Each X_i is multiplied by its corresponding pool. The result is then accumulated at the output vector Y_i using the adders and the feedback lines as shown in Figure 5.11.

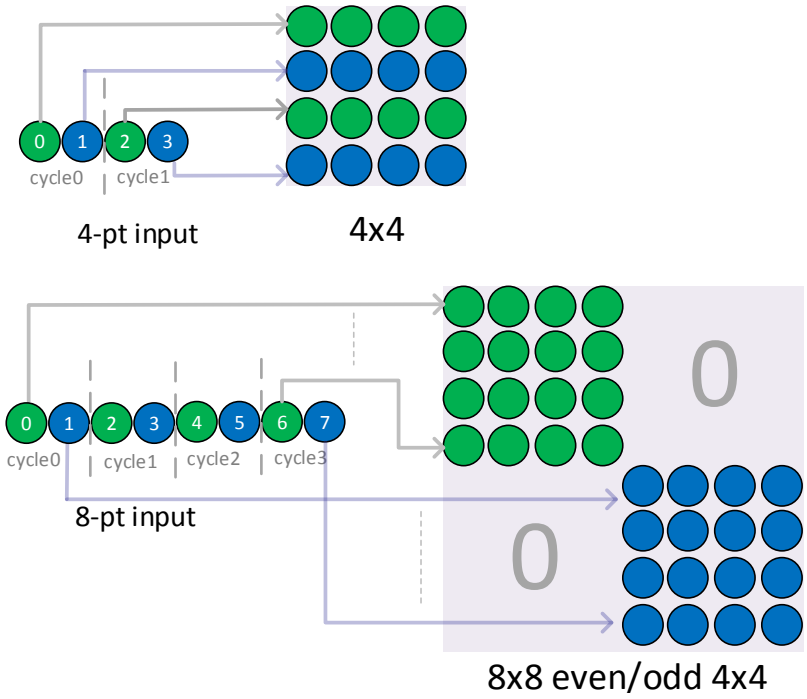


Fig. 5.10 Multiplication pools for 4-point and 8-point iDCT-2

5.4 Experimental and Synthesis Results

VHDL hardware description language is used to implement both proposed designs. A state-of-the-art logic simulator [66] is used to test the functionality of the transform module. The test strategy is as follows. First, a set of 10^5 pseudo-random input vectors have been generated and used as test patterns. Second, a software implementation of the transforms has been developed, based on the transform procedures used in latest VTM [48]. Using self-check techniques, the bit accurate test-bench compares the simulation results with those obtained using the reference software implementation. The proposed design supports three different video standards including Advanced Video

Table 5.4 Synthesis results for both ASIC and Field Programmable Gate Array (FPGA) platforms

		CM arch.	RM arch.
ASIC	Frequency (Mhz)	450	450
	Combinational area (cell area)	221659	59588
	Noncombinational area (cell area)	27860	32413
	Buf/Inv area (cell area)	16796	4848
	Total area (cell area)	266315	96849
FPGA	Frequency (Mhz)	139	165
	ALMs (/427200)	51182 (12%)	9723 (2%)
	Registers	16924	14368
	DSP blocks(/1518)	0 (0%)	32 (2%)

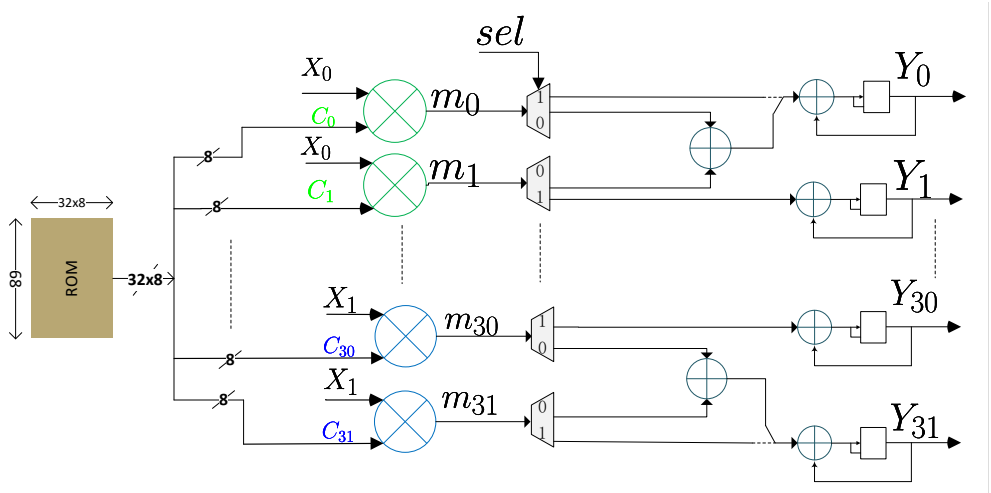


Fig. 5.11 RM architecture, X_0 and X_1 are the input samples, C_i is the transform coefficient, m_i is a multiplier and Y_i represents the output

Coding (AVC)/H.264, High Efficiency Video Coding (HEVC)/H.265 and the emerging VVC/H.266 standard. The CM 1-D architecture MTS core works at 450Mhz and achieves a total of 249K cell area, while the RM architecture achieves a total 93K cell area and 17408 bits of ROM used to store transform coefficients. Both modules has been synthesised using Design Compiler (DC) of synopsys with Taiwan Semiconductor Manufacturing Company (TSMC) 28nm standard cell library. Table 5.4 gives the synthesis results for both architectures on ASIC and Arria 10 FPGA platforms. It shows that the RM architecture consumes 63% less gates than the CM one. A fair comparison with other

Table 5.5 Comparison of different hardware transform designs

Solutions	Fan *et al.* [63]	Garrido *et al.* [59]	Mert *et al.* [57]	Kammoun *et al.* [61]	Proposed RM architecture	
Technology	ASIC 65 nm	20 nm ME	ASIC 90 nm	ME 20 nm FPGA	ASIC 28 nm	ME FPGA
Gates/ALMs	496400	1312	417000	133017	96849	9723
Registers	–	3624	–	–	–	14368
DSPs	–	32	–	1561	–	32
Frequency (Mhz)	250	458.72	160	147	600	165
Throughput (fps)	–	3840 × 2160p23	7680 × 4320p39	1920 × 1080p50	3840×2160p30	1920×1080 p50
Memory	–	41 × 21 Kbits	–	–	–	
Transform size	4, 8, 16, 32	4, 8, 16, 32	4, 8	4, 8, 16, 32	4, 8, 16, 32, 64	
Transform type	DCT-2/8, DST-7	DCT-2/8, DST-7	DCT-2/8, DST-7	DCT-2/5/8, DST-1/7	DCT-2/8, DST-7	

studies in the literature is quite difficult. Most studies focus on earlier version of VVC, and there are few ASIC-based designs for the VVC MTS. For comparison, Table 5.5 lists the key performance of state-of-the-art ASIC and FPGA-based works, including the VVC-MTS related works [57,59,61,63]. Gate count is the logical calculation part and it can be seen from Table 5.5 that compared with implementations of Fan *et al.* [63] and Mert *et al.* [57], our solution has obvious advantages. We present a unified transform architecture that can compute iDCT-2/iDST-7/iDCT-8 for transform unit of order 4,8,16,32 and 64 with a fixed throughput of 2 pixels per cycle. Practically, up to 80.5% of the hardware area can be reduced compared to [63] and up to 76.7% compared to [57]. In term of Adaptive Logic Modules (ALMs), we provide 92.7% reduction compared to implementation proposed by

Kammoun *et al.* in [61]. However, currently, no implementation have been found for the iDCT-2 of order 64. Although, [63] and [61] supports 2D for all transform types, the transform could be achieved only up to the 32×32 block size.

5.5 Conclusion

The design of the MTS hardware implementation involving the DCT-2/8, DST-7 transformation types has been studied in this chapter. In terms of hardware architecture, two aspects were studied. A CM-based and RM-based implementations of these transforms for size 4,8,16, 32 and 64 were presented. The first one is based on eliminating the multiplication operations by replacing them with adders and shifts. The second uses 32 multipliers in a a shared design for all transform types and sizes. This investigation showed that the RM-based architecture consumes 63% less area compared to the CM-based one. For VVC ASIC decoder, the RM-based architecture was adopted since it enables a significant area saving The 1D MTS design allows for 4K video decoding at 30 frames per second for 4:2:0 color sub-sampling. To the best of our knowledge, this is the first MTS implementation design that takes into account all types and asymmetric blocks.

The VVC transform involves more than one tool in its process, the Low Frequency Non-Separable Transform (LFNST) is one of the most critical tools of this process. Taking into consideration the conclusion we reached from the MTS study, the LFNST process should be implemented using RM-based architecture. The unified hardware design that jointly supports the MTS and LFNST is investigated in the next chapter.

Chapter 6

HW LFNST and MTS inverse transform design for 4K VVC Decoders

6.1 Introduction

In this work, the aim is to design a real time Versatile Video Coding (VVC) video decoding chip that supports 4kp120 video resolution with a target frequency of 450Mhz. The Application-Specific Integrated Circuit (ASIC) chip is expected to decode the target resolution over two cores. As a result, each core is expected to process 4Kp60 resolution in real time. As described in the previous chapter, since the transform modules process residuals in $N \times N$ pixel units, it is convenient to use their number per second to specify and measure the throughput.

Table 6.1 Low Frequency Non-Separable Transform (LFNST) & 1-D Multiple Transform Selection (MTS) throughput performance per block size for 4Kp60:420 resolution @450Mhz

Unit size	Throughput (unit/s)	Clock budget (cycles)	DCT-2 (samples/cycle)	DST-7/DCT-8 (samples/cycle)	LFNST (samples/cycle)
64x64	182250	1238	1	✗	4
64x32,32x64	364500	619	2	✗	4
64x16,16x64	729000	310	2	✗	4
64x8,8x64	1458000	155	2	✗	4
64x4,4x64	2916000	78	2	✗	4
32x32	729000	310	4	1	4
32x16,16x32	1458000	155	4	2	4
32x8,8x32	2916000	78	4	2	4
32x4,4x32	5832000	39	4	2	4
16x16	2916000	78	4	4	4
16x8,8x16	5832000	39	4	4	4
16x4,4x16	11664000	20	4	4	4
8x8	11664000	20	4	4	4
8x4,4x8	23328000	10	4	4	4
4x4	46656000	5	4	4	4

Table 6.1 shows the detailed performance for each block size for the inverse 1-D MTS and inverse LFNST. It highlights the cycle budget available for each input block and the required number of processed residuals per cycle according to the target frequency. As we can notice from Table 6.1, the required throughput for the 1-D iMTS and iLFNST is 4 samples/cycle. Both modules were designed so that they can use the same input and output memories. Figure 6.1 depicts the top level hardware architecture that regroups the separable and non-separable transforms as depicted in folded architecture. The 1-D MTS is expected to process 4 samples/cycle. As a result, we get a mean throughput of 2 samples/cycle for the 2-D MTS. Since the inverse LFNST share the same input/output memories with the 1-D iMTS, it follows the same throughput of 4 samples/cycle as shown in Figure 6.1. To further enhance the chaining between transform blocks, the modules should sustain a fixed system latency. This enables an exact prediction of the VVC transform performance. The transform modules follow the latency of the 64×64 transform block, regardless of the input block size.

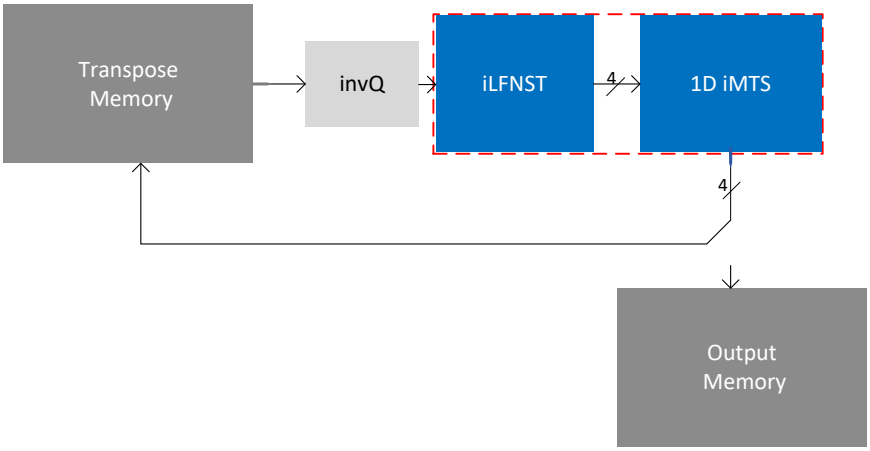


Fig. 6.1 2D VVC transform in a folded architecture

The LFNST + MTS processor control interface is summarized in Table 6.2. A positive pulse in *input_enable* launches the transform process, with the block size defined by *tr_width* and *tr_height*. The MTS and LFNST control signals are defined in the interface with prefix $MTS\_..$ and $LFNST\_..$, respectively. For example, the transform type and direction for the MTS are defined in the *MTS_type* and *MTS_dir* input signals, respectively. Following the *input_enable* state, the tr_in signal will carry four N_{bi} samples at the next clock cycle. N_{bi} and N_{bo} are respectively the bit size of the input and output sample, they depend on the bit depth. These samples are then fed to the LFNST module. The result, carried by *LFNST_out*, is then sent to the MTS module, which will perform the second dimensional transform. After finishing the 2-D transform, the MTS communicates the inverse transformed samples to the output memory via *MTS_out_fin* signal. However, if the first direction is selected for the MTS, the results will be directed to the input memory via *MTS_out_inter*. For both modules that integrate data loading and pipeline stages, the design starts generating the results of the inverse LFNST/1-D MTS after a fixed system latency. It then generates thanks to a pipelined architecture the outputs every clock cycle without any stall.

This chapter tackles the problem of hardware implementation of the inverse 1D VVC MTS and

Table 6.2 LFNST & MTS design interface.

Signals	I/O	#BITS	Description
clk	I	1	System Clock
rst_n	I	1	System reset, active low
input_enable	I	1	Activation pulse to start
tr_width	I	3	Transform Block width: 000:4 - 100:64
tr_height	I	3	Transform Block height: 000:4 - 100:64
MTS_type	I	2	Tran. type: DCT-2, DCT-8 and DST-7
MTS_dir	I	1	Transform direction : 0: Hori.; 1: Vert.
LFNST_pos_x	I	5	top left x Position
LFNST_pos_y	I	3	top left y Position
LFNST_set_idx	I	2	LFNST kernel set index
LFNST_idx	I	2	LFNST kernel index
tr_src_in	I	$4N_{bi}$	LFNST input data
MTS_ready	O	1	Ready pulse, end of N-point MTS
LFNST_ready	O	1	Ready pulse, end of LFNST
MTS_out_inter	O	$4N_{bi}$	Intermediate output, 1-D MTS
MTS_out_fin	O	$4N_{bo}$	Final output, 2-D MTS
LFNST_out	O	$4N_{bi}$	LFNST output data

LFNST targeting 4Kp60 resolution and 450Mhz frequency. Both, the 1-D iMTS and iLFNST should sustain a fixed system latency with a fixed throughput of 4 samples/cycle. Based on our previous study [67], the best approach to implement the iMTS is thought an Regular Multiplier (RM)-based architecture. It is also the same approach used to implement the iLFNST. In this work, we present two hardware architectures for the VVC iMTS and iLFNST using an RM-based architecture. The following sections are organized as follows. The proposed hardware design for both iMTS and iLFNST RM-based architectures are described in Section 6.2 and Section 6.3, respectively. Section 6.4 discusses the unified design of the iLFNST and iMTS. Section 6.5 presents the experimental setup and the synthesis results of the VVC transform implementations with a comparison with the High Efficiency Video Coding (HEVC) transform. Finally, Section 6.6 concludes the chapter.

6.2 1-D iMTS design

Similar to the previous iMTS study, this design uses three main modules to perform the 1-d transformation. However, this design should support a 4 samples/cycle throughput which is double the throughput of the previous design. Thanks to the scalability of the first design, the new module has the same top level architecture but with certain upgrades. These upgrades impact mainly two modules that were described in sections 5.3.1 and 5.3.3:

6.2.1 Read Only Memory

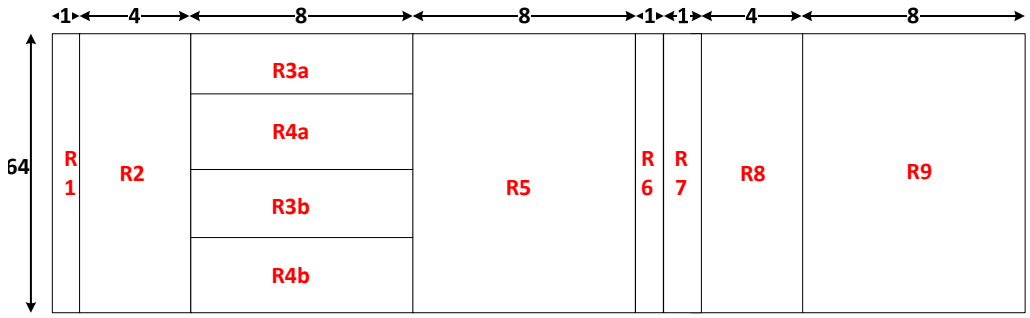


Fig. 6.2 Read Only Memory for 4 samples/cycle iMTS decomposed over nine regions

For the 4 samples/cycle, all matrices are stored in the Read-Only Memory (ROM) as shown in Figure 6.2. The latter is of height 64-bits (4×16) and of width 35. Two 16×16 matrices of the iDiscrete Cosine Transform (DCT)-2 are spread over 4 regions and stored one on top of the other in regions: 3a, 3b, 4a and 4b. These latter is sided by 32×16 matrix of the 64×64 odd decomposition. The other iDiscrete Sine Transform (DST)-7 transform sizes are stored side by side in regions from 6 to 9. The regions within the memory are as follows:

- R1: the 4×4 of 4-pt DCT-2 and the 4×4 odd decomposition of the 8-pt DCT-2 matrices.
- R2: the 16×16 DCT-2 matrix.
- R3a: the even rows of the 16×16 DCT-2 matrix.
- R4a: the even rows of 16×16 odd decomposition of the 32×32 DCT-2 matrix. .
- R3b: the odd rows of the 16×16 DCT-2 matrix
- R4b: the odd rows of 16×16 odd decomposition of the 32×32 DCT-2 matrix.
- R5: the 32×16 odd decomposition of the 64×64 DCT-2 matrix
- R6: the 4×4 DST-7 matrix.
- R7: the 8×8 DST-7 matrix.
- R8: the 16×16 DST-7 matrix.
- R9: the 32×16 DST-7 matrix.

This division enables to efficiently handle the MTS rate, by keeping one memory access for every 4 samples. As shown previously, the ROM height is 64 coefficients where each one is coded over 8-bits. Therefore, the total memory size is 17920-bits which is 35 columns of 512-bit (64 × 8-bits) each (35×512-bits). Finally, a 6-bit address bus is sufficient to cover the entire memory space.

6.2.2 Multiplication Module

The multiplication module is the core module of the iMTS and thanks to the sample rate it uses a total of 64 multipliers. The transform coefficients, retrieved from the ROM, are feed to the core multiplier with the four input samples as shown in Figure 6.3. X_0, X_1, X_2, X_3 in the Figure represent the input samples while $C_0, C_1 \ldots C_{63}$ are the MTS coefficients. Like the first design, the multiplication process is applied over multiple pools. The iMTS uses four pools each holds a total of sixteen coefficients and assigned to one single input sample. The final transformed result is delivered through the output Y interface at a throughput of 4 samples/cycle.

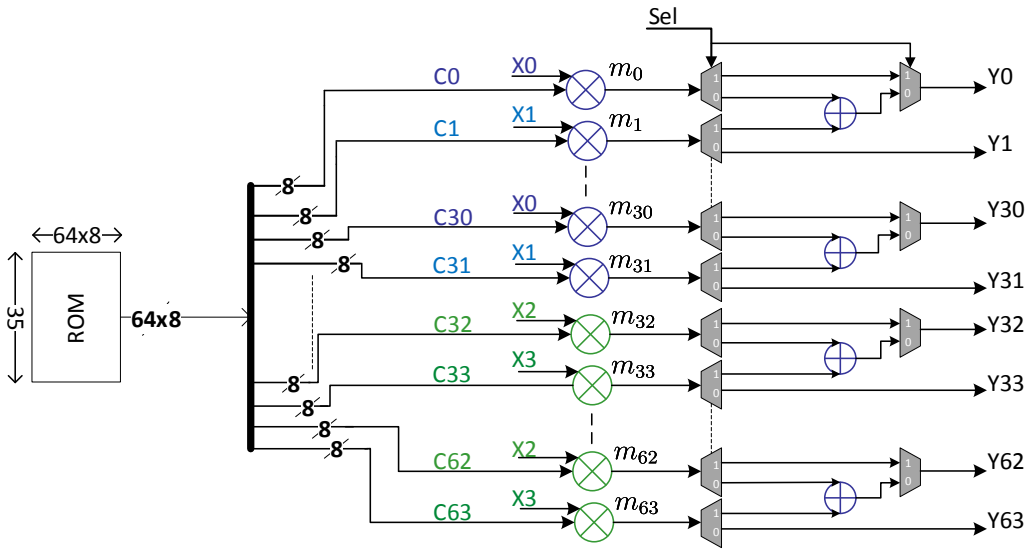


Fig. 6.3 RM architecture, $X_0, X_1, X_2 and X_3$ are the input samples, C_i is the transform coefficient, m_i is a multiplier and Y_i represents the output.

Table 6.3 summarizes the required multipliers and input/output rates for every transform block.

Table 6.3 Summary table for iMTS functional consumption

Type	Size	Input rate (samples/cycle)	Output rate (samples/cycle)	used multipliers	Butterfly (On/Off)
0 (DCT-2)	4-point	4	4	16	off
	8-point	4	4	16	on
	16-point	4	4	64	off
	32-point	4	4	64	on
	64-point	2	4	64	on
1/2 (DST-7/DCT-8)	4-point	4	4	16	✗
	8-point	4	4	32	✗
	16-point	4	4	64	✗
	32-point	2	4	64	✗

6.3 iLFNST design

The inverse LFNST module is an element of the transform chain pipeline that operates on dequantized coefficients and performs an inverse secondary transform, whose output is a set of transform coefficients. The inverse LFNST can be summed up by the matrix multiplication in Eq (6.1):

$$Y = M \cdot X, \tag{6.1}$$

where Y is the resulting vector (16 or 48 coefficients) and M is the transform matrix (size 16×8, 16×16, 48×8 or 48×16, depending on block size) and X is the input vector (8 or 16 coefficients) resulting from a reorganization of the input block matrix into a vector. The top level architecture of the iLFNST module is depicted in Figure 6.4.

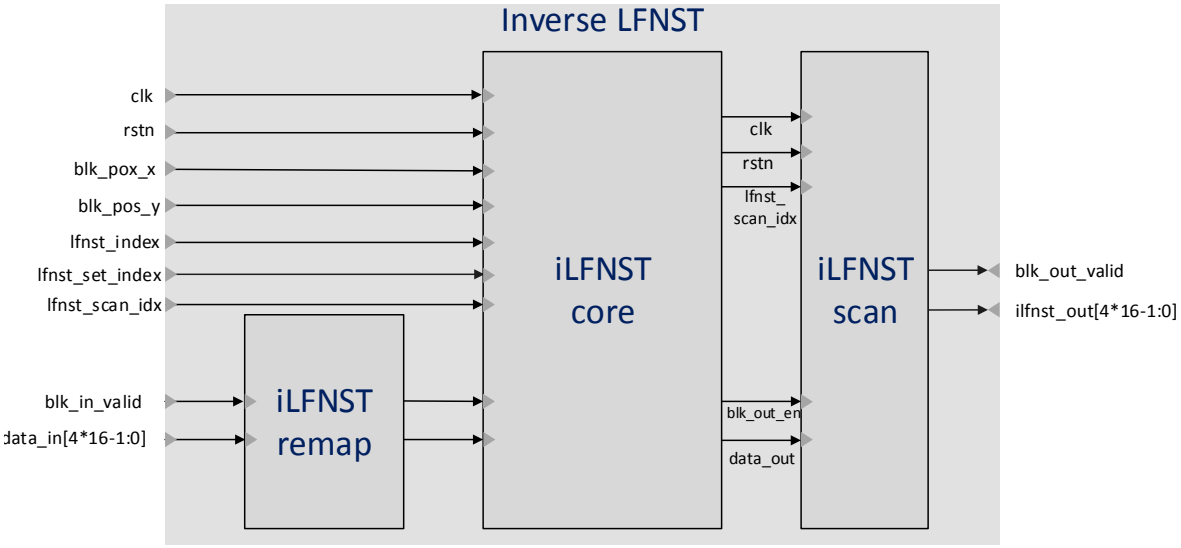


Fig. 6.4 iLFNST top level design

The module receives four input samples (each 16 bits) associated with a valid signal. Alongside the data, some associated parameters are received: The block coordinates x and y (top sample of the input vector) the block's LFNST set index, index and scan index derived based on the intra-prediction mode as shown in Chapter 4.

The samples are processed through three sub-modules:

- The ilfnst_remap process remaps the input block onto a vector.
- ilfnst_core performs iLFNST transform on the remapped vector.
- ilfnst_scan maps the data from a vector to a block.

6.3.1 iLFNST remap module

The remapping module is in charge of mapping the input samples presented on the input according to the order illustrated in Figure 6.5. In respect to the throughput of the iLFNST module, the remapping

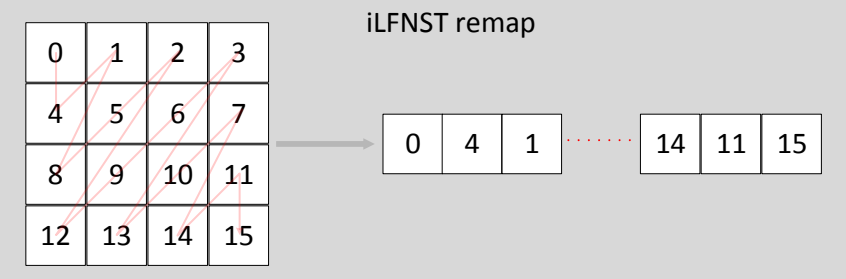


Fig. 6.5 iLFNST remapping process

process is performed over four cycles.

6.3.2 iLFNST core module

The inverse LFNST core will perform the transform using a matrix multiplication as shown in Figure 6.6:

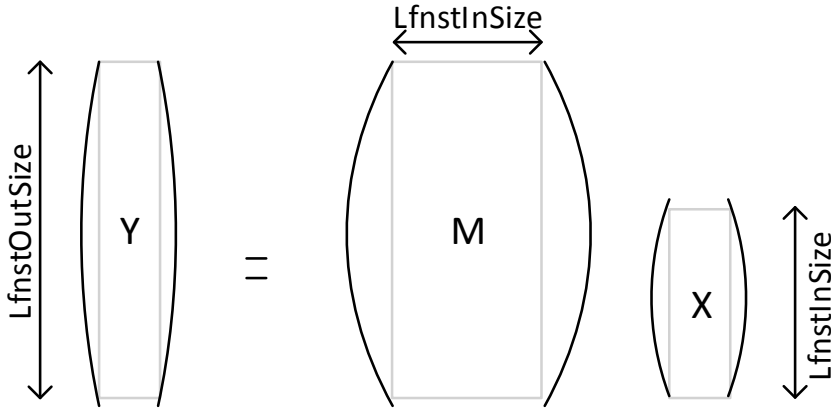


Fig. 6.6 LFNST matrix multiplication

Where the LFNST output size ($LfnstOutSize$) could have a maximum size of 48 and the LFNST input size ($LfnstInSize$) a maximum size of 16. Table 6.4 sums up the input output and matrix characteristics for each block size.

Table 6.4 LFNST input output characteristics

Block size	Input size	Output size	Matrix size
4×4	8	16	8×16
4×N, N×4	16	16	16×16
8×8	8	48	8×48
M×N, N, M >8	16	48	16×48

For our system, the desired throughput is 4 samples/cycle. The critical path of our design is when the input size and the output size equals 16. For the other cases, the input/output rates can be

separated to maintain 4 samples/cycle at the output. For instance, for the 4×4 block the input rate can be decreased to 2 samples per 1 clock cycles. For 8×8 block sizes, it can be decreased to 1 sample for every three clock cycles. For $M \times N$ and $N \times M$ (M, N>8) blocks, the input rate is set to 2 samples per three clock cycles. In all cases, an output rate of 4 samples/cycle is maintained. Table 6.5 summarizes the input output rate for each block size along with the required multipliers.

Table 6.5 input/output rate table

Block size	Input rate (samples/cycle)	Output rate (samples/cycle)	clock budget (cycles)	required multipliers
4×4	2	4	4	32
4×N, N×4	4	4	N	64
8×8	1/3	4	N	22
M×N, N, M >8	2/3	4	N	44

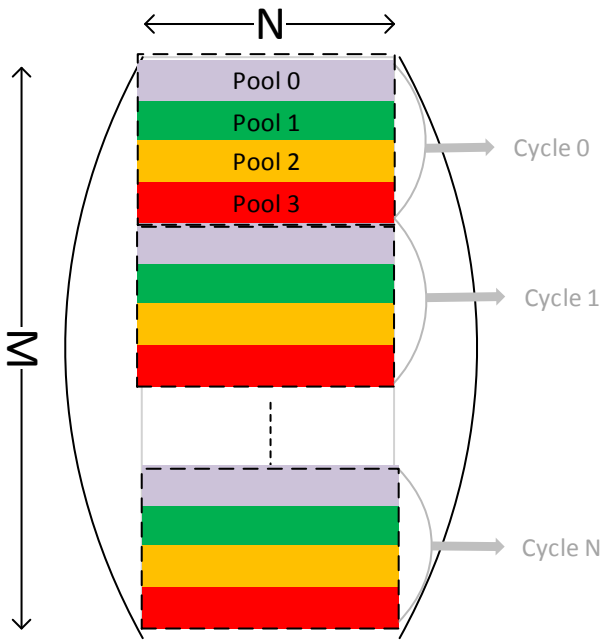


Fig. 6.7 iLFNST multiplication pools. Four pools every clock cycle associated to one input sample.

As can be seen, 64 RM are needed to perform the iLFNST. They are split into four pools of 16 multipliers each. At each cycle, each pool is associated and holds a certain part of the transform matrix, as can be seen in the Figure 6.7 (each colour being associated to one pool at one clock cycle). N and M are respectively the LFNST kernel dimensions (M = 16 or 48 and N = 8 or 16).

The iLFNST core module is composed of three sub-modules similar the iMTS as depicted in Figure 6.8. First, the ROM module in which all the LFNST kernels are stored. Second, the coefficient preparation module where, depending on the LFNST set index and LFNST index, the required LFNST kernel are retrieved. Third, the core multiplication module in which the inverse LFNST transform is applied by multiplying the data input and the selected matrix, and finally, the transformed outputs are

delivered to the *ilfsnt_scan* sub-module through the *ilfnst_out* interface.

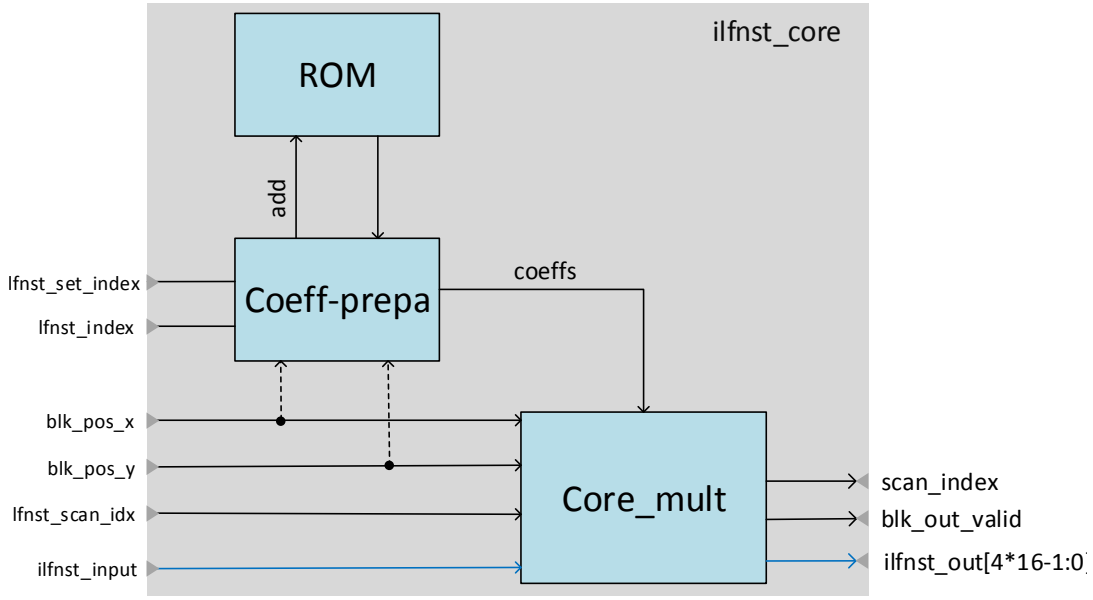


Fig. 6.8 iLFNST core modules

6.3.2.1 ROM

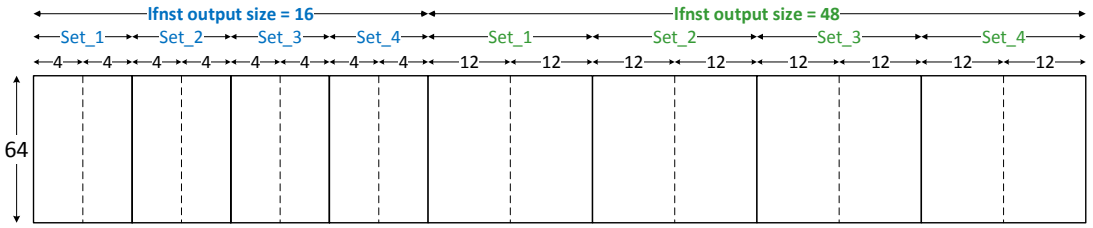


Fig. 6.9 iLFNST Read Only Memory

The ROM stores the LFNST kernels with respect to the scanning order. The first 4×64 region in the ROM corresponds to the first kernel of size 16×16 (lfnst_set_index = 1 and lfnst_index = '0' and lfnst block size out = 16). The first 64 coefficients (for ROM address = 0) are the first and second lines of the 16×16 matrix. The following 64 coefficients of ROM address = 1 are the fourth and the seventh lines of the 16×16 matrix. The remaining kernels are stored in the same way in the ROM. This fact that the kernels are stored in a mixed fashion allows us to skip the kernel remaps module and directly call the core multiplication module. The kernel elements are stored over 8-bits which makes the total memory size of 65536-bits, which is 128 in width and 512-bits (512=64×8) depth.

Table 6.6 Coefficient preparation module input/output mapping

Input			Output	
LFNST output size	LFNST set index	LFNST index	ROM address	data retrieved
16	1	0	0x0 to 0x3	16×16 matrix 0
	1	1	0x4 to 0x7	16×16 matrix 1
	2	0	0x8 to 0xB	16×16 matrix 2
	2	1	0xC to 0xF	16×16 matrix 3
	3	0	0x11 to 0x14	16×16 matrix 4
	3	1	0x15 to 0x18	16×16 matrix 5
	4	0	0x19 to 0x1C	16×16 matrix 6
	4	1	0x1D to 0x20	16×16 matrix 7
48	1	0	0x21 to 0x2C	48×16 matrix 0
	1	1	0x2D to 0x38	48×16 matrix 1
	2	0	0x39 to 0x44	48×16 matrix 2
	2	1	0x45 to 0x50	48×16 matrix 3
	3	0	0x51 to 0x5C	48×16 matrix 4
	3	1	0x5D to 0x68	48×16 matrix 5
	4	0	0x69 to 0x74	48×16 matrix 6
	4	1	0x75 to 0x80	48×16 matrix 7

6.3.2.2 Coefficient preparation module

The coefficients preparation module is an in-between module that selects the appropriate set of coefficients from the ROM, and delivers them to the multiplication module. Using the LFNST set index, the LFNST index and the LFNST output size, the coefficients preparation module decides which region is accessed using what address. Table 6.6 shows the mapping between the input state and the ROM addresses and the retrieved data.

6.3.2.3 Core multiplication

Figure 6.10 shows the core multiplication module of the LFNST at 4 samples/cycle. It uses a total of 64 multipliers spreed over four pools of 16 coefficients size each. In the Figure, C represents the coefficient matrix retrieved from the ROM memory. X_0, X_1, X_2, X_3 are the 4 input samples captured at a random clock cycle. Z is an intermediate vector which holds 48 value representing the maximum size of largest LFNST output non zero size (48). Y_0, Y_1, Y_2, Y_3 are the 4 output samples containing the results of the inverse LFNST transform. The $S1$ depicts the accumulation phase which is applied after performing the multiplication. This phase depends on the input block size. For each input/output size combination, a number of additions will be applied. The result will be accumulated to the appropriate Zi with a total number of accumulations that depends on the input size. The $S2$ stage outputs the final results at a rate of 4 samples per cycle.

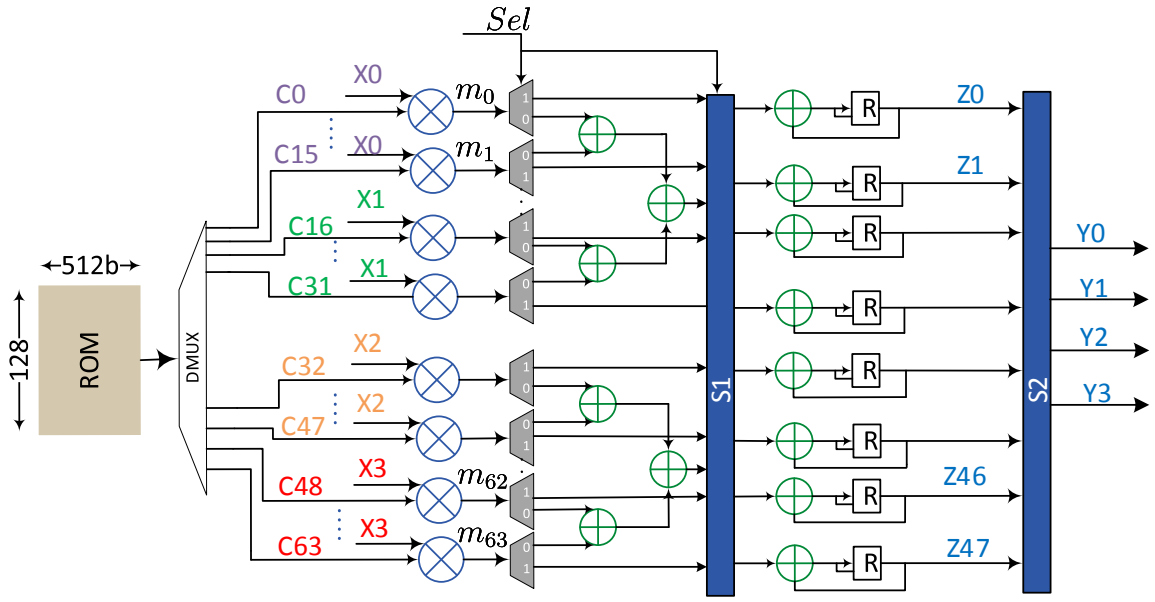


Fig. 6.10 LFNST core multiplication @4 samples/cycle

6.3.3 iLFNST scan module

At iLFNST core output, the input data are organized as a 16 or 48 samples vector. This latter will be mapped to form a block of size 4×4 or 8×8 for 16 and 48 input vectors, respectively. The output organization is performed as shown in Figure 6.11 which depends on the LFNST scan index (derived from the Intra prediction mode) and the input block size.

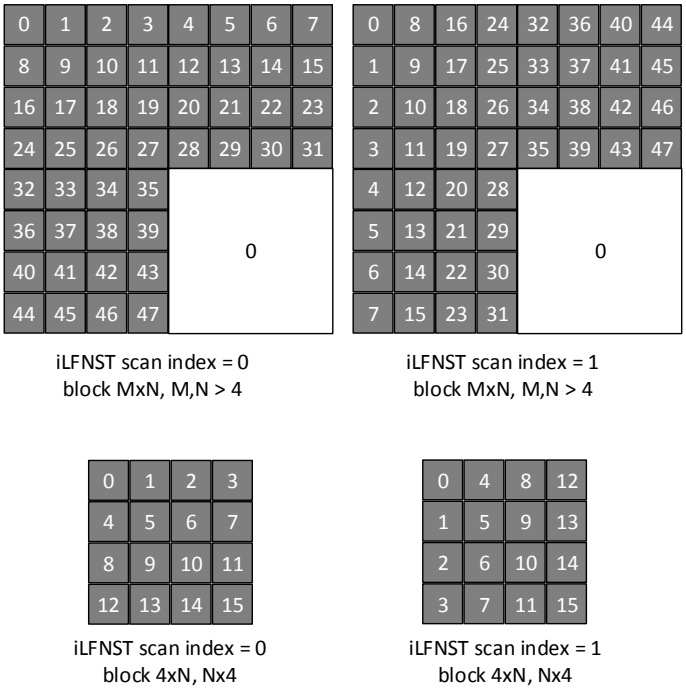


Fig. 6.11 Scanning iLFNST core output data depending on iLFNST scan index and input block size

6.4 Unified inverse LFNST + MTS design

Figure 6.12 illustrates the proposed hardware architecture for inverse MTS and LFNST modules. The first control unit, shown in Figure 6.12:(b), outputs two selection signals *sel1* and *sel2* which are derived based on the transform type *mts_type* (ie. DCT-2 or DCT-8/DST-7) and the direction *mts_dir* (ie. vertical or horizontal). If the transform type (*mts_type*) refers to DCT-2, the input samples are processed by the inverse LFNST module, otherwise they go through the *Bypass* module.

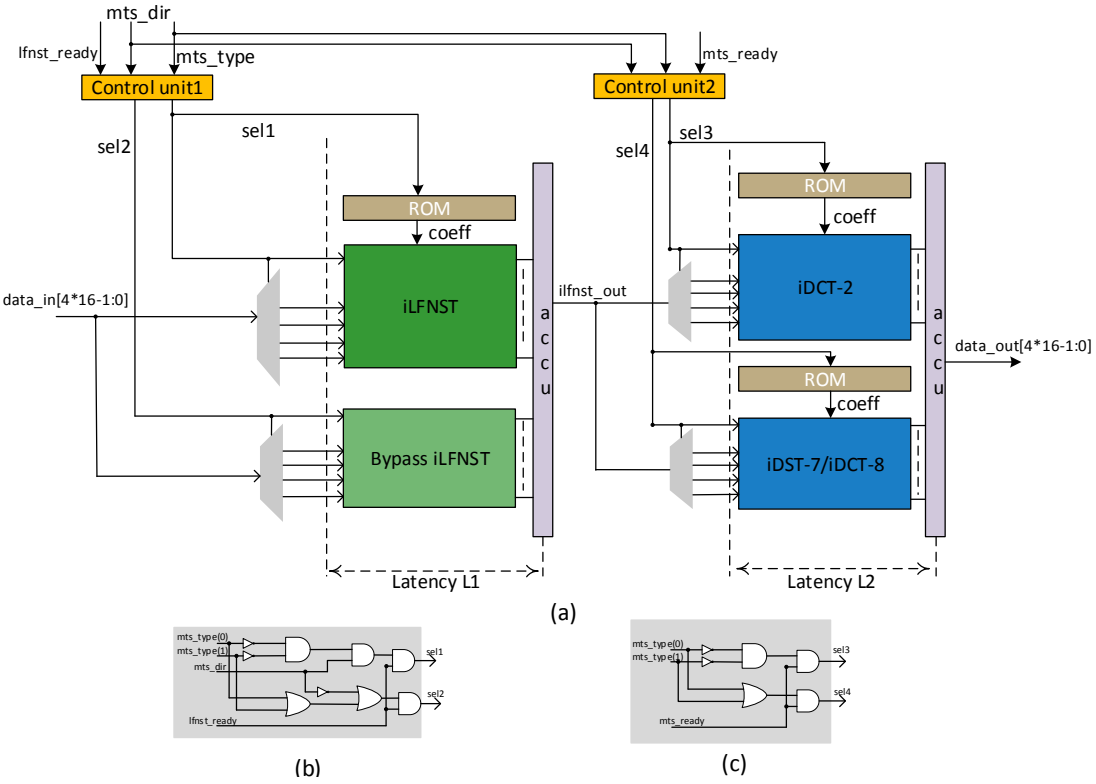


Fig. 6.12 Proposed hardware iMTS and iLFNST architecture, (a) Unified design for all sizes and all transform types, (b) Control unit 1 for iLFNST, (c) Control unit 2 for iMTS.

The inverse LFNST module uses 64 multipliers in a shared architecture for all block sizes and kernels. Both inverse LFNST and *bypass* modules have the same latency *L1*, for that, the *Bypass* module uses registers to create a delay line with latency *L1*. At the end of the inverse LFNST, the output samples are accumulated and delivered to the iMTS modules in a 4 samples/cycle rate. The second control unit, shown in Figure 6.12:(c), delivers two signals *sel3* and *sel4*. These signals are derived from the transform type *mts_type*. *sel3* enables the DCT-2 module, while *sel4* enables DCT-8/DST-7 one. These two modules share 64 multipliers and have the same latency *L2*. At the end of the transformation, four output samples are delivered through *data_out* signals every clock cycle.

Figure 6.13 depicts the VVC transform (MTS + LFNST) pipeline. Together they consume a total of 128 Regular Multipliers m_j $j \in \{0,\ldots,127\}$. As can be seen, 64 for the iMTS $j \in \{0,\ldots,63\}$ and 64 for the iLFNST $j \in \{64,\ldots,127\}$. The LFNST is performed before the MTS and can retrieve from its ROM, depending on the LFNST index and set index, the 64 coefficients that correspond to

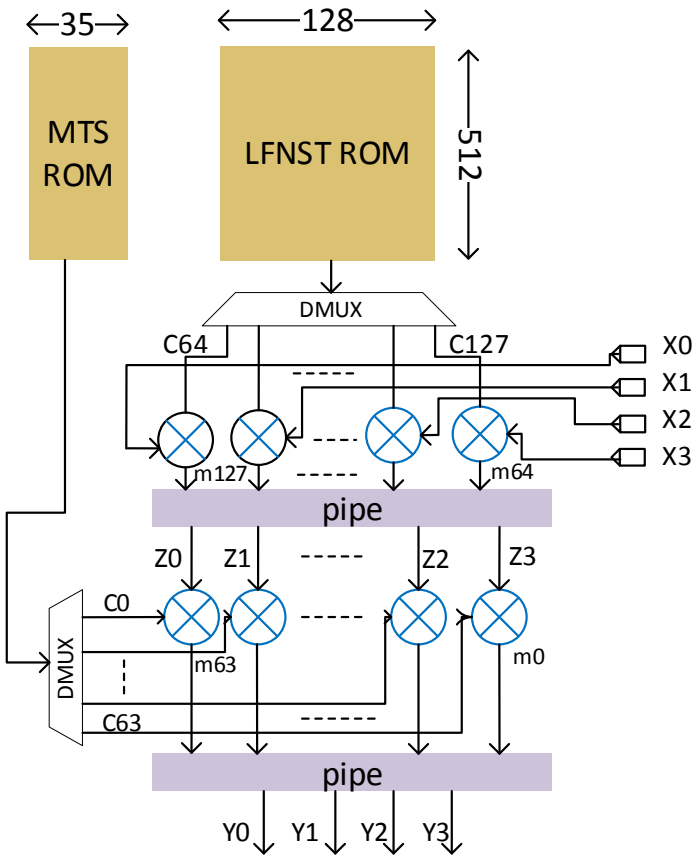


Fig. 6.13 Pipeline the VVC transform module including iMTS and iLFNST. Four samples X_i are processed every clock cycle

the four input residuals shown as C_j with $j \in \{32,\ldots,63\}$. These coefficients are then feed to the core multipliers along with the input samples X_0, X_1, X_2 and X_3 at a random clock cycle. The result of the multiplication process is forwarded to the delay line which is depicted by the pipe of Figure 6.13. This latter ensures a constant system latency for the iLFNST. At the end of the iLFNST processing, the output samples are feed the iMTS module with a rate of 4 samples/cycle through Y_0, Y_1, Y_2 and Y_3 signals. These samples are then fed to the core multipliers along with the corresponding coefficients retrieved from the iMTS ROM C_j with $j \in \{0,\ldots,63\}$. The final result goes to the second delay line which unifies the latency for the iMTS. Finally, the transformed samples are redirected to either the transpose or output memory via Z_i signals. They are directed to the transpose memory if the first direction is selected for the MTS and to the output memory otherwise.

6.5 Experimental and Synthesis Results

6.5.1 Experimental setup

VHSIC Hardware Description Language (VHDL) hardware description language is used to describe the proposed transform block. A state-of-the-art logic simulator [66] is used to test the functionality

of the VVC transform module (MTS + LFNST). The design was synthesised by a commercial Design Compiler [68] using 28-nm technology. The test strategy is performed as follows. First a set of 10^5 pseudo-random input vectors has been generated and used as test patterns to test the 1-D MTS in unit tests. The same number of tests was used to verify the LFNST module. Second, a software implementation of the inverse MTS and LFNST has been developed, based on the transform procedures defined in the last version of the VVC Test Model (VTM)12 [4]. Using self-check technique, the bit accurate test-bench compares the simulation results with those obtained using the reference software implementation. The tests cover all transform block sizes from 4×4 to 64×64 including asymmetric blocks, for both iMTS and iLFNST modules.

6.5.2 Coding performance

First the coding gain and software complexity of the MTS and LFNST modules under the VVC common Test Conditions in three main coding configurations including All Intra, Random Access and Low Delay B are analysed [69]. Table 6.7 gives the coding losses and the complexity reductions at both encoder and decoder when the MTS and LFNST tools are turned off. It can be noticed that disabling MTS and LFNST tools introduces the highest coding losses in All Intra coding configuration with 1.32% and 0.99% Bjøntegaard Delta Rate (BD-BR) [70, 71] losses when these two tools are disabled, respectively. The MTS and LFNST tools are active only for Intra blocks to limit the complexity overhead at the encoder side. This may explain the higher coding gain brought by the MTS and LFNST in All Intra configuration compared to the two other Inter configurations. Moreover, the MTS tool significantly increases the encoder complexity since five sets of horizontal and vertical transforms are tested to select the best performing set of transforms in terms of rate distortion cost. At the decoder side, the MTS slightly increases the decoder complexity since only one transform set is processed by block. This slight complexity overhead is mainly caused by introducing more complex transforms including DST-7, DCT-8 and DCT-2 of size 64. Disabling the LFNST tool changes the encoder behaviour resulting in complexity increase at the encoder side caused by complexity reduction tools included in the VTM to speedup the encoder such as early termination [72, 73]. In fact, the LFNST tool enables more efficient coding of residuals resulting in better energy packing and more coefficients are set to zeros after the quantisation. Therefore, the early termination techniques integrated in the VTM [72] terminate the Rate Distortion Optimization (RDO) process earlier when the LFNST is enabled resulting in lower encoding time. At the decoder side, disabling the LFNST slightly decreases the decoder complexity since LFNST is applied only on blocks processed by the DCT-2 primary transform. This study shows that both MTS and LFNST have a slight impact on the software decoder complexity. In the next section, the impact of both MTS and LFNST tools will be investigated on a hardware decoder targeting an ASIC platform.

Table 6.7 Performance (%) in terms of BD-BR and run time complexity when MTS and LFNST tools turned off in VTM8.0 [3]. Evaluations performed under the VVC common test conditions [4].

Disabled tool	All Intra Main 10					Random Access Main 10					Low Delay B Main 10				
	Y	U	V	*EncT*	*DecT*	Y	U	V	*EncT*	*DecT*	Y	U	V	*EncT*	*DecT*
MTS	1.32	0.96	1.02	85	99	0.75	0.60	0.53	89	95	0.53	0.26	0.04	93	96
LFNST	0.99	1.98	2.21	110	100	0.70	0.78	1.08	96	100	0.33	0.88	0.96	107	98

6.5.3 Synthesis Results

6.5.3.1 iMTS performance

The proposed 1-D MTS design was synthesized using Design Compiler (DC) targeting an 28-nm ASIC at 450 MHz. The design sustain 4 samples/cycle with a fixed system latency. The total area consumed by the 1-D MTS is 138.9 Kgates. The 2-D MTS is applied using the 1-D core in a folded architecture, where the output of the first direction is redirected to a transpose memory which re-sends the transposed data to the same 1-D core. As a result the 2-D MTS includes both 1-D MTS and the transpose memory. The same architecture can be found in many state of the art works, however the proposed MTS design supports all sizes and types of VVC transform including the block of order 64 for the DCT-2. The first design of the MTS proved to be scalable enabling to double the performance of the design from a throughput of 2 samples/cycle to a throughput of 4 samples/cycle. The flexibility of the design allowed this transition to be done only by doubling the total number of multipliers from 32 to 64 multipliers while conserving the same architecture. Table 6.8 shows the synthesis results for both design, the 4 samples/cycle design increases the area cost by 55% instead of 100% and this is because doubling the number of multipliers reduces the size of the delay line which is not negligible compared to the total area size.

Table 6.8 Synthesis results for MTS-2 p/c and 4 p/c at 450 MHz.

		MTS-2p/c	MTS-4p/c
ASIC 28-nm	Num. of mult.	32	64
	Combinational area	59135	97210
	Non-combinational area	29946	41705
	Total area (gate count)	89082	138916

Table 6.9 Synthesis results for LFNST-2 p/c and 4 p/c at 450 MHz.

		LFNST-2p/c	LFNST-4p/c
ASIC 28-nm	Num. of mult.	32	64
	Combinational area	49911	79409
	Noncombinational area	24680	29806
	Total area (gate count)	74592	109215

6.5.3.2 iLFNST performance

Like the MTS, the proposed LFNST design was synthesized using DC targeting an 28-nm ASIC at 450 MHz. Because it is chained to the 1-D MTS it sustains the same throughput of 4 samples/cycle with a fixed system latency. The total area consumed by the LFNST design is 109.2 Kgates. Unlike the MTS, the LFNST is a new tool introduced in the VVC standard. That is why no study was found in literature that investigates a hardware implementation of LFNST. As a result, this work is the first study for a hardware architecture for the LFNST. Similar to MTS, the scalability of LFNST design enabled a very flexible transition going from a throughput of 2 samples/cycle to a throughput of 4 samples/cycle. The design proved to be scalable, doubling the throughput comes only at the expense of doubling the total number of multipliers while the same architecture is preserved. Table 6.9 shows the synthesis results for both designs, the 4 samples/cycle design increases the area cost by 46.5% instead of doubling it and this is because the delay line is reduced. It is noteworthy that the percentage of area increase for the LFNST is lower than for the MTS. This is caused by the delay line of the LFNST which is larger than in the MTS.

6.5.4 Inverse VVC Transform compared to inverse HEVC Transform

To compare the VVC transform and the HEVC transform, we synthesised two 1-D design that are similar in throughput. The 1-D HEVC transform supports all DCT-2 and DST-7 for block size 4×4, it has a throughput of 2 samples/cycle with a static system latency. The VVC transform supports all MTS types and sizes and all LFNST sizes with a throughput of 2 samples/cycle and a fixed system latency. The difference between the two transforms is that the VVC one is designed based on a RM architecture while HEVC transform is designed using a Constant Multipliers (CM) based architecture. Many studies proved that using CM based architecture for HEVC transform is more efficient in terms of area than using RM based one. However, based on our previous study in [67], it turns out that for VVC using RM based architecture is more area efficient due to the new transform types and sizes. This fact proves that both architectures are optimally designed and thus giving more credibility for fair comparison. Table 6.10 shows the performance results of both designs. It can be noticed that the

Table 6.10 VVC vs HEVC Transform synthesis results at 450 MHz.

		VVC DCT-2+DCT-8+DST-7+LFNST	HEVC DCT-2+DST-7
ASIC 28-nm	Architecture	RM	CM
	Total area	163672	41500

area of VVC transform supporting both separable and non-separable transforms is 4 time larger than the HEVC transform block, this is considered to be reasonable rather good due of the new tools and new sizes introduced in VVC. If only the separable transform is considered, we notice from Table 6.11 that VVC separable transform is around 2 times larger than the HEVC one. This is because the separable transform of VVC supports larger block sizes up to 64×64 and uses a new transform type such as DCT-8 and DST-7.

Table 6.11 VVC vs HEVC separable Transform synthesis results at 450 MHz.

		VVC MTS (DCT-2+DCT-8+DST-7)	HEVC DCT-2+DST-7
ASIC 28-nm	Architecture	RM	CM
	Total area	89082	41500

6.6 Conclusion

In this chapter a hardware implementation of the inverse VVC transform module has been investigated targeting a real time VVC decoder on ASIC platforms. The VVC transform module consists of a primary transform block and a secondary transform block called MTS and LFNST, respectively. The proposed hardware implementation relies on regular multipliers and sustains a constant system latency with a fixed throughput of 4 samples per cycle. The MTS design leverage all primary transform optimisations including butterfly decomposition for DCT-2, zeroing for 64-point DCT-2 and 32-point DST-7/DCT-8, and the linear relation between DST-7 and DCT-8. The proposed design uses 128 RMs (64 for MTS and 64 for LFNST) enabling to reach real time decoding of 4K video (4:2:0) at 60 frames par second. The proposed transform design has been successfully integrated in a hardware ASIC decoder supporting the transform module of recent MPEG standards including Advanced Video Coding (AVC), HEVC and VVC. To the best of our knowledge, this is the first hardware investigation that presents a study for a unified design between the LFNST and MTS inverse transform module.

Chapter 7

Transform HW design studies and optimizations for 4K VVC encoder

7.1 Introduction

In the previous chapters, we presented a full study for the Versatile Video Coding (VVC) transform module (Multiple Transform Selection (MTS)+Low Frequency Non-Separable Transform (LFNST)) in the context of the decoder. In that context, the VVC transform is computed directly using information retrieved from the bitstream. It applies the inverse MTS and inverse LFNST transform blocks by fetching the transform type and LFNST indices from the bitstream. However, in the encoder context the approach is different: the types and the indices of the transform modules are defined as a result of an exhaustive search to derive the best Rate Distortion (RD) candidate which is then signaled in the bit-stream. RD candidates for a given block size are actually variations of transform encoding parameters which are put in competition by comparing their RD costs. The lowest the RD cost the better the candidate. In High Efficiency Video Coding (HEVC), the partitioning decision was based on comparing all possible quad-tree combinations of a Coding Tree Unit (CTU). In VVC, the number of partitioning combinations has exploded due to the new Binary Tree (BT) and Ternary Tree (TT) patterns and the use of asymmetric blocks at the transform level and the new transform type combinations. In order to avoid processing a huge number of computations in the full search mode, the encoder must defines methods to reduce and compensate the complexity of the search. The best choice is to perform a full search with a limited number of Multi Type Tree (MTT) possibilities with the objective of finding a good trade-off between the encoding efficiency and hardware feasibility. Finally, for every block size the encoder runs all possible combinations to find the best RD candidates for the transform as shown in Figure 7.1.

As we can see, the MTS tests five possible combinations between the horizontal and vertical direction: (Discrete Cosine Transform (DCT)-2, DCT-2), (DCT-8, Discrete Sine Transform (DST)-7), (DCT-8, DCT-8), (DST-7, DCT-8) and (DST-7, DST-7). The LFNST on the other hand tests two possible indices where each index represents an LFNST kernel: LFNST K0 and LFNST K1. This latter is tested only when the DCT-2 is enabled for the MTS. Along with primary and secondary transforms combination, the transform skip scenario is also tested as a candidate for the encoder

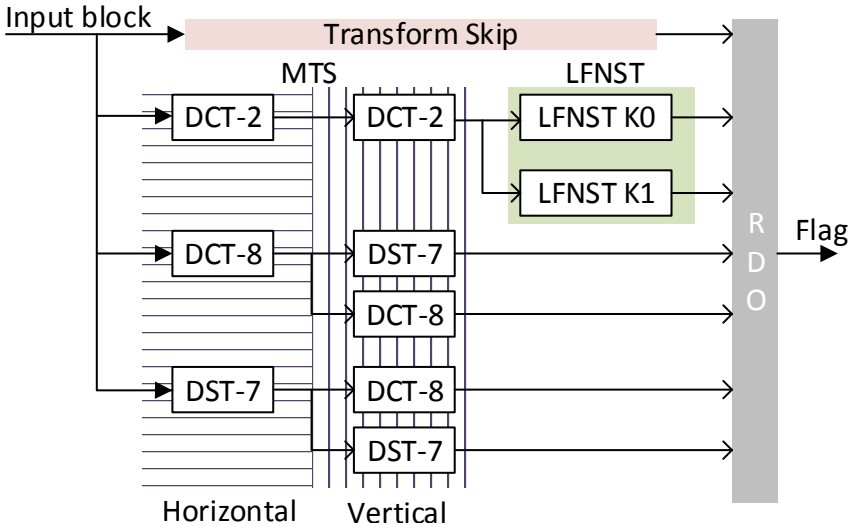


Fig. 7.1 All combinations search for the encoder LFNST + MTS transforms

transform.

In this work, the VVC encoder is expected to encode a real time 4Kp60 videos over four cores where each core runs 1080p60 videos at a frequency of 450Mhz. To do so, it defines a fixed budget for every 64×64 CTU. This budget is uniform for the four cores and it equals to *9000* clock cycles. This is the main constraint that defines the processing power of the transform loop. Furthermore, in order to reduce the complexity of full search, the block partitioning is limited to one BT split only and the transform candidates selected for inter and intra modes are diversified. Tables 7.1 and 7.2 show the transform RD candidates for each block size for intra and inter modes, respectively. As we can notice, the tables highlight the number of candidates for Luma and Chroma components, the total number of samples to be processed and the required samples per cycle with respect to cycle budget allocated to the the 64×64 CTU. The idea is to compute the total number of samples to be processed in a CTU for: all block sizes, all color components, all RD candidates. The total number of samples is then divided by the CTU cycle budget to get an estimation of the required parallelism (number of samples to be processed per cycle). It can be seen that the average throughput for the transform pipeline for I frames is 95 samples/cycle while for P/B frames is set to 84 samples/cycle.

Tables 7.1 and 7.2 show also that the number of RD candidates for P/B frames is less than the one for I frames. In fact, for I frames, the encoder tests eight configurations for the MTS and LFNST combined while for P/B frames it tests only seven candidates (Cand 1 is not tested). In the encoder context, a transform chain represents the sequence of forward transform, forward quantisation, inverse transform and inverse quantisation. In order to reduce complexity, one DCT-2 chain is dropped when P/B frames are being processed. The configurations of the LFNST and MTS for I frames are as follows:

- Cand 0: Transform skip
- Cand 1: DCT-2 for the two directions
- Cand 2: DCT-2 for the two directions with LFNST kernel index 0

Table 7.1 Transform candidates and total samples area for I frames in respect to 64 × 64 CTU budget

	Samples/block	Luma			Chroma 422			
		Y Blocks	Y RD cond	Y Area	UV Blocks	UV RD cond	UV Area	Total YUV area
CU 64×64	4096	1	8	32768	1	8	32768	65536
CU 64×64 BT	4096	2	8	65536	2	8	65536	131072
CU 32×32	1024	4	8	32768	4	8	32768	65536
CU 32×32 BT	1024	8	8	65536	8	8	65536	131072
CU 16×16	256	16	8	32768	16	8	32768	65536
CU 16×16 BT	256	32	8	65536	32	8	65536	131072
CU 8×8	64	64	8	32768	64	8	32768	65536
CU 8×8 BT	64	128	8	65536	128	8	65536	131072
CU 4×4	16	256	8	32768	256	8	32768	65536
							total area	**851968**
							samples/cycle	**94,66***

$* = \frac{total\ area}{ctu\ buget}$

Table 7.2 Transform candidates and total samples area for P/B frames in respect to 64 × 64 CTU budget

	Samples/block	Luma			Chroma 422			
		Y Blocks	Y RD cond	Y Area	UV Blocks	UV RD cond	UV Area	Total YUV area
CU 64×64	4096	1	7	28672	1	7	28672	57344
CU 64×64 BT	4096	2	7	57344	2	7	57344	114688
CU 32×32	1024	4	7	28672	4	7	28672	57344
CU 32×32 BT	1024	8	7	57344	8	7	57344	114688
CU 16×16	256	16	7	28672	16	7	28672	57344
CU 16×16 BT	256	32	7	57344	32	7	57344	114688
CU 8×8	64	64	7	28672	64	7	28672	57344
CU 8×8 BT	64	128	7	57344	128	7	57344	114688
CU 4×4	16	256	7	28672	256	7	28672	57344
							total area	**745472**
							samples/cycle	**82,83***

$* = \frac{total\ area}{ctu\ buget}$

- Cand 3: DCT-2 for the two directions with LFNST kernel index 1
- Cand 4: DCT-8 for the horizontal direction and DCT-8 for vertical direction
- Cand 5: DCT-8 for the horizontal direction and DST-7 for vertical direction
- Cand 6: DST-7 for the horizontal direction and DST-7 for vertical direction
- Cand 7: DST-7 for the horizontal direction and DCT-8 for vertical direction

For I frames the average samples rate required is 95 samples/cycle while for P/B frames the rate is 83 samples/cycle. In practice, large data buses are problematic for hardware design due to the limitation of memory interfaces. Therefore, a single transform chain loop is not the appropriate solution and then it is more suitable to separate the transform loop on multiple transform chains where each chain deals with only 32 samples/cycle in average. This solution provides some kind of flexibility while maintaining a uniform low bus size for the entire transform loop. As a result, a total of three transform chains for I and P/B frames separately are used. Figure 7.2 shows the transform chains for I and P/B frames. As we can see, for intra/inter frames, the CTU cycles is shared between 3 dedicated transform chains, each one processing 32 samples per cycle. Since one DCT-2 candidate

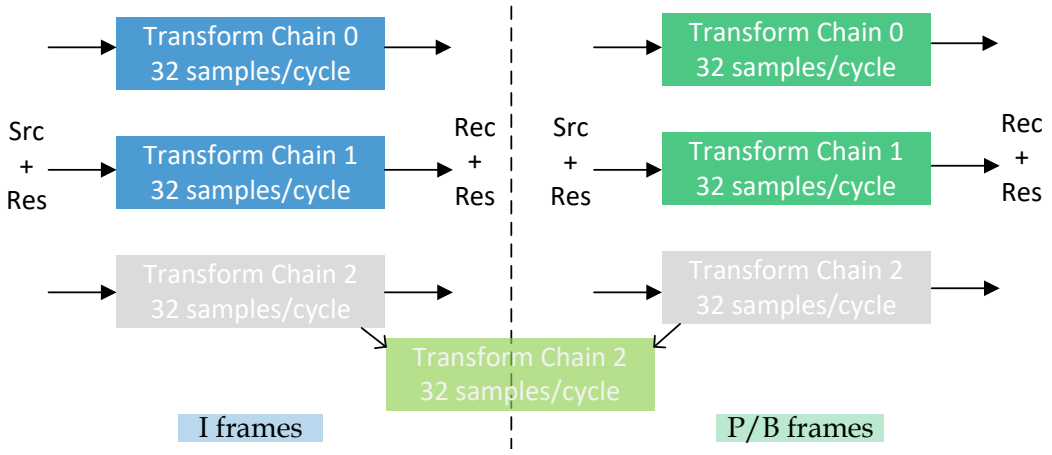


Fig. 7.2 Transform chains for I and P/B frames

(Cand 1 is not tested) is dropped for P/B frames, it is convenient to plug the additional intra DCT-2 candidate within a P/B transform chain and share some of its resources with an intra chain as shown in Figure 7.2. In fact, there are two ways to plug the additional DCT-2: the first one is to plug it with one of the LFNST cores and the second one is to plug it with its inverse DCT-2. This latter is the subject of this chapter.

In this chapter, we present a comparison between a shared design uniting the DCT-2 and its inverse and a shared design uniting the DCT-2 and the LFNST at a throughput of 32 samples/cycle. The following sections are organized as follows. The cost of each shared design separately is studied in Section 7.2 and Section 7.3, respectively. Section 7.4 presents a comparison between both solutions and highlights the main differences. Finally, Section 7.5 concludes the chapter.

7.2 Shared LFNST/DCT design for 32 pixels/cycle throughput

This section presents a study for the shared LFNST/DCT-2 core. The LFNST is applied as an secondary transform. For an encoder, it operates between the primary transform and the quantization processes. Different from the primary transform, the secondary transform is only operated on the low-frequency transform coefficients. It only processes the top-left corner of the TB and is only applied when DCT-2 is selected for the primary transform. On the other hand, DCT-2 is a separable transform and applied for all TB sizes and can be only reduced for block size 64 using the zeroing as described in the background chapter. The aim of this study is to evaluate the performance of shared design between the LFNST and the DCT-2 cores for the purpose of resource sharing. In the transform chain, this shared design will exploited as depicted in Figure 7.3.

Figure 7.3 shows in blue the shared modules between the LFNST and the DCT-2. As we can see there are two main channels: one for I frame and a second for P/B frames. The LFNST is only applied on I frames while the DCT-2 is applied on P/B frames. For the purpose of resource sharing, two shared modules for the LFNST/DCT are proposed. These modules use the DCT-2 kernels (horizontal and vertical directions) when P/B frames are received and the LFNST/iLFNST kernels when I frames are received. After the first multiplexers, the input frame is delivered to the first module, which is a

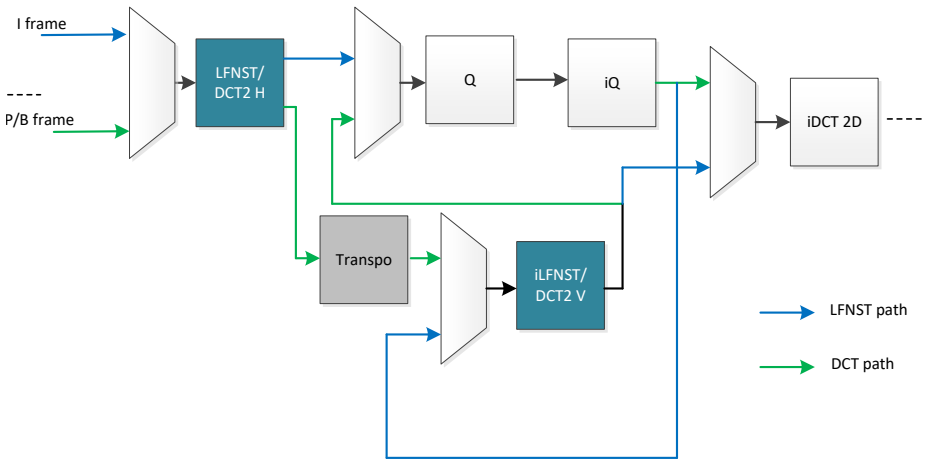


Fig. 7.3 Shared LFNST/DCT-2 design within the transform chain

shared architecture between the forward LFNST and 1-D horizontal DCT-2. If the input frame is an I frame, the data follows the LFNST path (in blue), otherwise it follows the DCT-2 one (in green). If the DCT-2 path is activated, the first module applies the DCT-2 of horizontal direction. Then the data is delivered to the second module (iLFNST/DCT-2V) through the intermediate transpose memory where the DCT-2 of second direction is applied. If LFNST path is activated, the module uses LFNST and iLFNST for the first and second design, respectively, and the data are delivered directly to decoder loop before applying the second module. A more functional diagram is depicted in Figure 7.4.

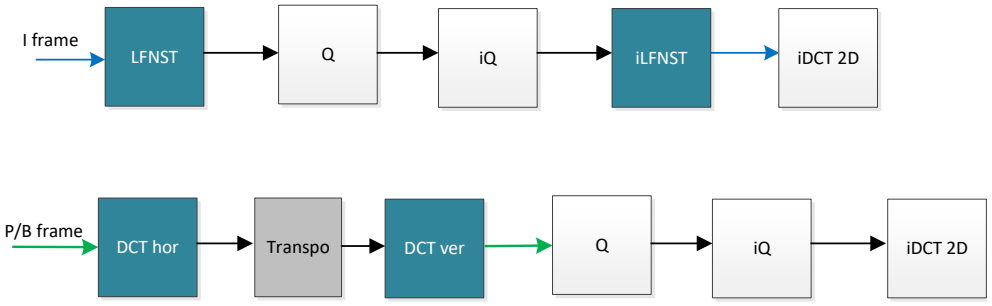


Fig. 7.4 Shared LFNST/DCT-2 design functional diagram

In this section, we evaluate the performance of these two modules in term of surface and number of multiplications. In addition, we describe the proposed hardware design for each in respect to the constraint of 32 pixels/cycle.

7.2.1 Shared design for forward DCT-2 horizontal and LFNST

The first module is a mutualization between the forward LFNST and the 1-D horizontal DCT-2. Table 7.3 shows the total number of multipliers required for this module. It shows also the number of multiplications needed per sample for different transform block sizes. As we can notice from Table 7.3, almost all input block sizes are measured based on the DCT-2 requirements. They are

Table 7.3 shared 1-D DCT-2 hor/forward LFNST multiplication cost

Forward LFNST			1-D DCT-2			cycle budget	total shared multipliers
Input block	Kernel size	Total Mult. for 32 samples/cycle	Input vector	Kernel size	Total Mult. for 32 samples/cycle	-	-
8×4	16×16	256	4	4×4	128	1	256
4×8	16×16	256	8	8×8	128	1	256
8×8	48×16	192	8	8×8	128	2	256
16×4	16×16	128	4	4×4	128	2	128
4×16	16×16	128	16	16×16	256	2	256
16×8	48×16	192	8	8×8	128	4	192
8×16	48×16	192	16	16×16	256	4	256
16×16	48×16	96	16	16×16	256	8	256
8×32	48×16	96	8	8×8	128	8	128
32×8	48×16	96	32	32×32	256	8	256
16×32	48×16	48	16	16×16	256	16	256
32×16	48×16	48	32	32×32	512	16	512
32×32	48×16	<48	32	32×32	512	32	512
16×64	48×16	<48	16	16×16	256	32	256
64×16	48×16	<48	64	64×64	1024	32	1024
32×64	48×16	<48	32	32×32	512	64	512
64×32	48×16	<48	64	64×64	1024	64	1024
64×64	48×16	<48	64	64×64	1024	128	1024
Total number of multipliers for 32 samples/cycle with DCT-2 of size 64							**1024**
Total number of multipliers for 32 samples/cycle without DCT-2 of size 64							**512**

bounded only by the DCT-2 performance. The worst-case scenario is found for DCT-2 of size 64, which requires a total of 1024 multipliers. However, if we eliminate the use of DCT-2 size 64, the consumption could be reduced by half and the total number of required multipliers is 512. Figure 7.5 depicts the proposed module architecture.

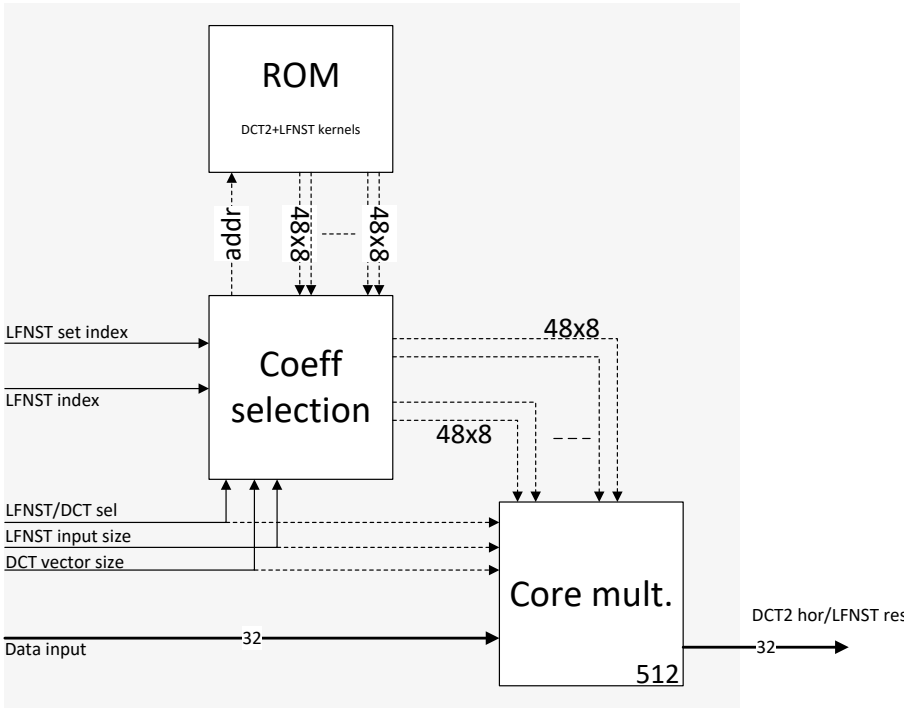


Fig. 7.5 Module block diagram for DCT-2 hor/forward LFNST

Based on our previous designs for the inverse LFNST and MTS, this module can be composed of three sub-modules:

- The Read-Only Memory (ROM): This module stores the DCT-2 and LFNST kernels.
- Coefficient selection: this module prepares the transform coefficients and delivers them to the next module. To process a 32 samples per cycle, it should communicate at maximum 1024 coefficients every clock cycle divided over several pools of maximum size 48 each.
- Core multiplication module: this module uses 1024 multipliers to transform input samples at 32 samples/cycle rate in case the size 64 block is supported otherwise it uses 512 multipliers.

7.2.2 Shared design for forward DCT-2 vertical and inverse LFNST

The second module is a mutualization between the inverse LFNST and the 1-D vertical DCT-2. Table 7.4 shows the total number of multipliers needed for this module. It highlights the number of multiplications required per samples for different transform block sizes and transform types. Same

Table 7.4 shared 1-D DCT-2 ver/Inverse LFNST multiplication cost

Inverse LFNST			1-D DCT-2			cycle budget	total shared multipliers
Input block	Kernel size	Total Mult. for 32 samples/cycle	Input vector	Kernel size	Total Mult. for 32 samples/cycle	-	-
8×4	16×16	256	4	4×4	128	1	256
4×8	16×16	256	8	8×8	128	1	256
8×8	16×48	64	8	8×8	128	2	128
16×4	16×16	128	4	4×4	128	2	128
4×16	16×16	128	16	16×16	256	2	256
16×8	16×48	32	8	8×8	128	4	128
8×16	16×48	32	16	16×16	256	4	256
16×16	16×48	32	16	16×16	256	8	256
8×32	16×48	32	8	8×8	128	8	128
32×8	16×48	32	32	32×32	256	8	256
16×32	16×48	16	16	16×16	256	16	256
32×16	16×48	16	32	32×32	512	16	512
32×32	16×48	<16	32	32×32	512	32	512
16×64	16×48	<16	16	16×16	256	32	256
64×16	48×16	<16	64	64×64	1024	32	1024
32×64	48×16	<16	32	32×32	512	64	512
64×32	48×16	<16	64	64×64	1024	64	1024
64×64	48×16	<16	64	64×64	1024	128	1024
Total number of multipliers for 32 samples/cycle with DCT-2 of size 64							**1024**
Total number of multipliers for 32 samples/cycle without DCT-2 of size 64							**512**

as the shared DCT-2 hor/forward LFNST, the worst-case scenario for this design is found for the DCT-2 of size 64, which requires a total of 1024 multipliers. However, if we eliminate the use of DCT-2 size 64 we can reduce the total number of multipliers by half. Figure 7.6 depicts the proposed hardware architecture.

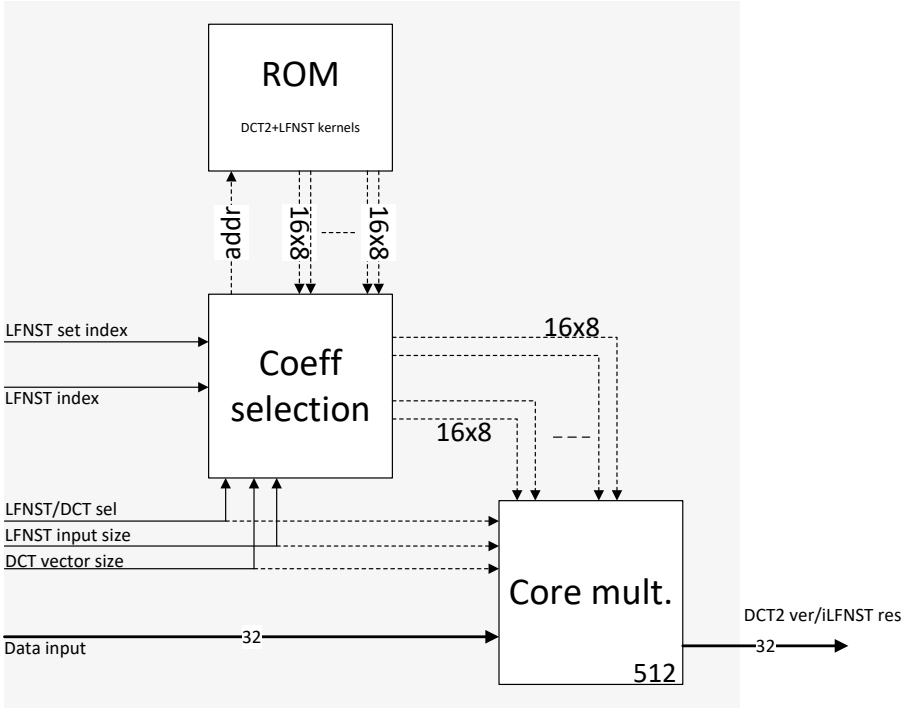


Fig. 7.6 Module block diagram for DCT-2 ver/inverse LFNST

As we can see, it can be composed of three main sub-modules:

- The ROM: this module stores the DCT-2 and LFNST kernels.

- Coefficient selection: this module prepares the transform coefficients and delivers them to the multiplication module. To process a 32 samples per cycle, it should communicate at maximum 1024 coefficients every clock cycle which are divided over several pools of maximum size 16 each. Each pool is associated to one input sample.

- Core multiplication module: this module uses 1024 multipliers to transform input samples at 32 samples/cycle rate in case the size 64 block is supported otherwise it uses 512 multipliers.

7.2.3 Discussion

This study has shown that for the two shared modules, the cost in terms of the number of multipliers, which in turn impacts the area of the circuit, is mostly measured on the consumption of the DCT-2 transform. The LFNST on the other hand uses much less multipliers than the DCT-2 especially with large input blocks. As a result, this approach may be inefficient. We could gain better by separating the LFNST from the DCT-2 and creating another shared architecture between the DCT-2 and its inverse while leaving the LFNST as independent module. This latter is the subject of the next section.

7.3 Shared DCT/iDCT design for 32 pixels/cycle throughput

The DCT-2 is a separable transform, it is applied for two direction (horizontal and vertical) as shown in Figure 7.7. At the encoder, the forward DCT-2 applies horizontal and vertical transforms for the first and second direction, respectively. On the other hand, at the decoder side, the inverse DCT-2 applies the vertical followed by the horizontal transform for the two directions. The 1-D DCT-2 transform is called N-point DCT (N represents the size of the input (line or column)).

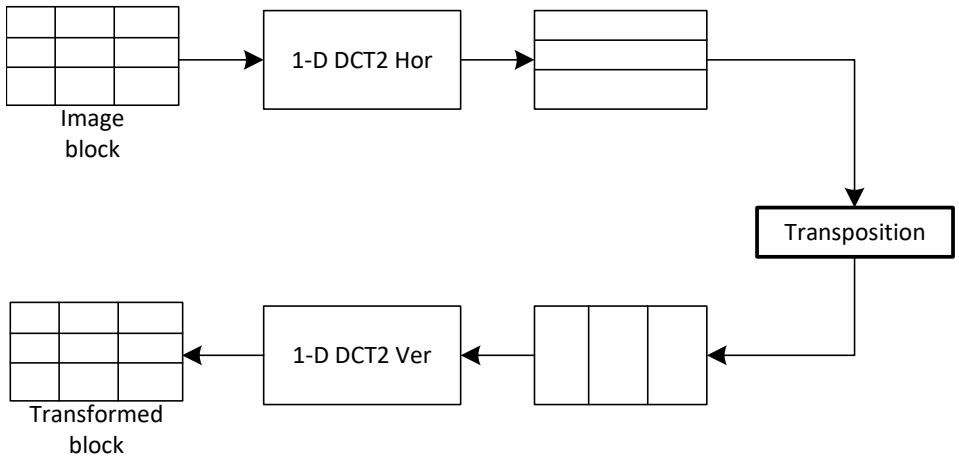


Fig. 7.7 2-D forward DCT-2

A 1-D DCT-2 core uses the same kernels regardless of the chosen direction. To create a shared design between DCT-2 and its inverse, the direction impacts only the pre-processing and post-processing stages. However, the core process is the same regardless of the selected direction.

Same as the previous shared design, the forward DCT-2 is an additional computation applied when P/B frames are transformed. The subject of this study is to evaluate the mutualisation of this core with the inverse DCT-2. Knowing that the iDCT-2 core is applied for another transform chain that is devoted for I frames. Thus the shared design should be as depicted in Figure 7.8.

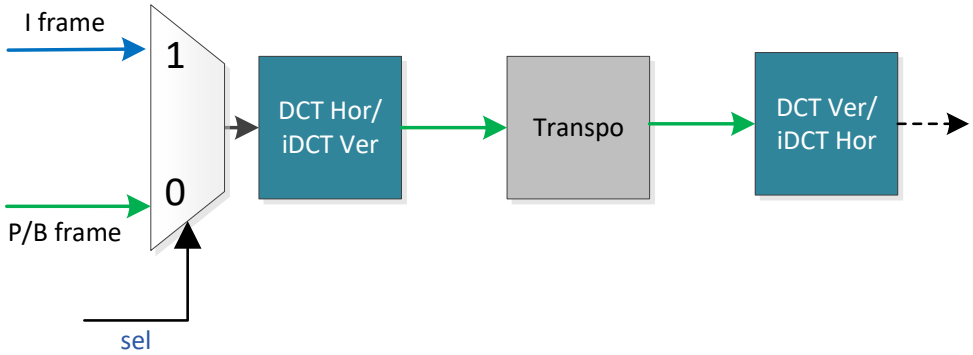


Fig. 7.8 Shared DCT/iDCT chain

Figure 7.9 depicts the proposed unified N-point DCT/iDCT circuit. As we can see, it is mainly composed of three parts: the prepossessing N-point butterfly for forward DCT, the core module and

the post-processing add/subtract module for the inverse DCT. When the forward DCT-2 is selected, the data follows the blue path in the Figure, it goes through the N-point butterfly and the core multiplication stages. However, when the inverse DCT-2 is selected, the data follows the green path where it goes through the core multiplication and add/sub stages.

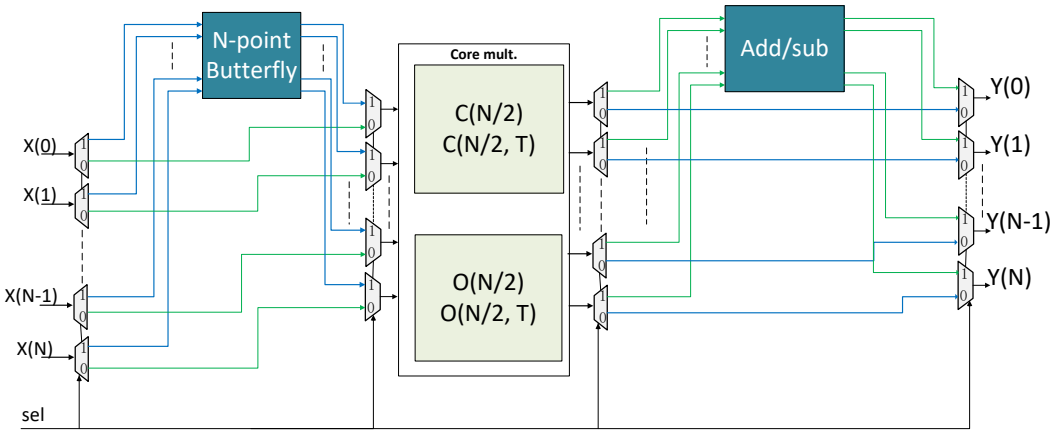


Fig. 7.9 Unified N-point DCT/iDCT architecture

7.3.1 N-point butterfly

This module is an array of "butterfly" circuits. It is applied when the forward DCT-2 is selected and it performs addition and subtraction operations on the input vector as shown in Eq (7.1):

$$\begin{aligned} a(i) &= X(i) + X(N-1-i) \\ b(i) &= X(i) - X(N-1-i) \\ where \quad & 0 \leq i \leq \frac{N}{2} - 1 \end{aligned} \tag{7.1}$$

The input samples X are delivered to the butterfly module when the DCT-2 is selected. The module then delivers a and b vectors as a result of the butterfly additions and subtractions stages. These two vectors will be later subject to core multiplication module. This module uses $N + \frac{N}{2} + \frac{N}{4} + ... + 4$ additions where N is the input block size. The maximum number of adders is 124 where N = 64.

7.3.2 Core multiplication

The core multiplication module applies the multiplication process for the DCT-2 and iDCT-2 using constant multipliers. The multiplication operations for an N-point DCT/iDCT are represented by the following equations:

- *For N-point DCT-2:*

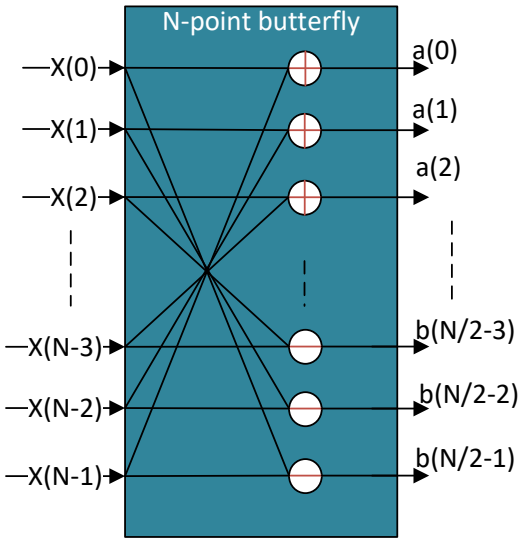


Fig. 7.10 N-point butterfly forward DCT-2 module

$$\begin{aligned} A' &= C_{N/2} \cdot a_{N/2} \\ B' &= O_{N/2} \cdot b_{N/2} \end{aligned} \tag{7.2}$$

The $x_{N/2}$ and $x'_{N/2}$ are the output of the butterfly module and they are connected to the input of the multiplication one. The A' and B' are the results of the core module and they are directed directly to the output interfaces. The kernels used for each N-point DCT are the even decomposition $C_{N/2}$ and the odd decomposition $O_{N/2}$ of the C_N transform matrix.

- *For N-point iDCT-2:*

$$\begin{aligned} A' &= C^T_{N/2} \cdot x_{i=0\ldots\frac{N}{2}-1} \\ B' &= O^T_{N/2} \cdot x_{i=\frac{N}{2}\ldots N} \end{aligned} \tag{7.3}$$

The $x_{N/2}$ and $x'_{N/2}$ are the input of the multiplication module. In this case the N-point butterfly module is bypassed. The A' and B' are the results of the core module and they are directed to the Add/sub stage before connected to the output interfaces. The kernels used for each N-point iDCT-2 are the even decomposition $C^T_{N/2}$ and the odd decomposition $O^T_{N/2}$ of the C^T_N inverse transform matrix.

- *Common features:*
  The similarity between the DCT and iDCT kernels is critical for the mutualization. The $O_{N/2}$ matrix is symmetric, i.e the transpose of the matrix is the same as the matrix. Therefore, the odd multiplication circuit can be used for both DCT and iDCT ($O^T_{N/2} = O_{N/2}$). This feature is not applicable for the even part ($C_{N/2}$), however, for the last decomposition of the even part

"C_2", we notice that the $C_2^T = C_2$. Thus, for every N-point DCT/iDCT circuit it is beneficial to reach the end of the butterfly decomposition (C_2, O_2) to fully mutualize the multiplication circuits. Figure 7.11 shows the example of 8-point DCT-2/iDCT-2 design. We can see that the same circuit is used when reaching the C_2/O_2 decomposition. This similarity between the kernels proves the advantage of mutualizing the DCT with its inverse. In the Figure, the blue

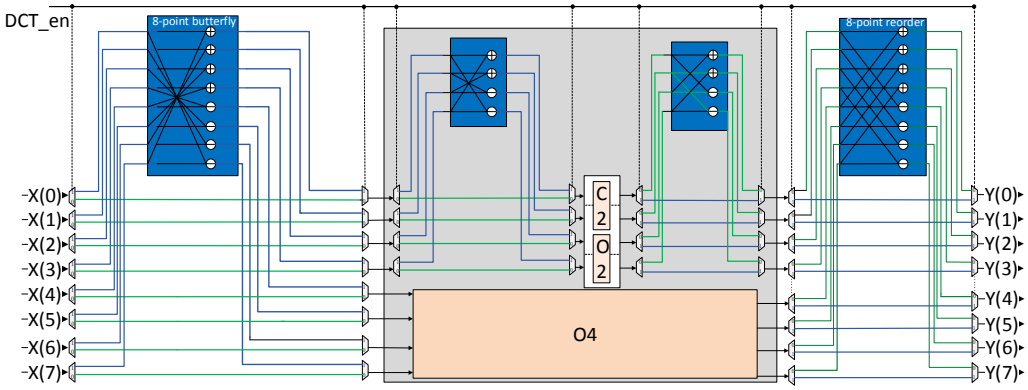


Fig. 7.11 8-point shared DCT-2/iDCT-2 design

lines represent the path of the DCT-2 data while the green line represents the iDCT-2 data path. We can notice that for both DCT-2 and iDCT-2 the same kernels are used ($C_2 = C_2^T$ and $O_N = O_N^T$). However, the mutualization adds a bunch of multiplexers at the input/output interfaces. The total number of multiplexers (TNM) required for each N-point DCT-2/iDCT-2 can be computed using the following equation: $TNM = \frac{32}{N} \times (N \times 4 + N/2 \times 4 + .. + 2 \times 4)$. For 8-point DCT-2/iDCT-2 (N = 8), the TNM = 224. It is important to note here that for the throughput of 32 samples per cycle, the circuit depicted in Figure 7.11 is duplicated four times in order to transform four columns/rows of 8 samples each in parallel.

7.3.3 Add/sub

The add/sub module is activated when inverse DCT-2 core is selected. This latter depicts the following equation:

$$\begin{aligned} Y_{i=0..\frac{N}{2}-1} &= A'_{N/2} + B'_{N/2} \\ Y_{i=\frac{N}{2}..N-1} &= A'_{N/2} - B'_{N/2} \end{aligned} \tag{7.4}$$

Where Y in this case represents the final transformed data, and A' and B' are the result of the multiplication module as depicted in Eq (7.5):

$$\begin{aligned} A'_{N/2} &= C^T_{N/2} \cdot X_{even} \\ B'_{N/2} &= O^T_{N/2} \cdot X_{odd} \end{aligned} \tag{7.5}$$

Figure 7.12 depicts this module.

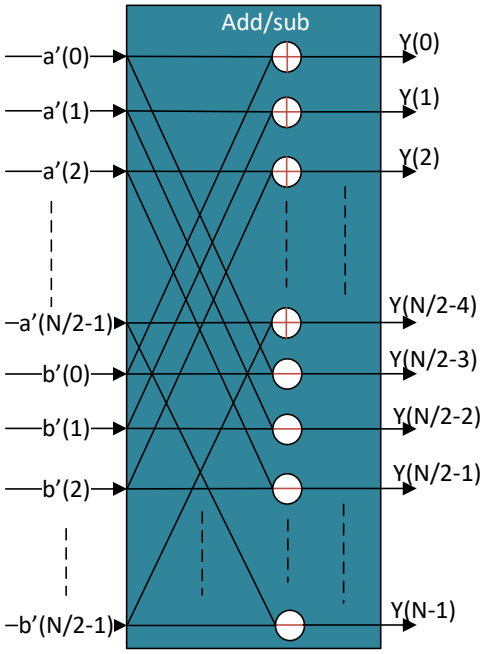


Fig. 7.12 N-point add/sub inverse DCT-2 module

7.3.4 Performance summary

The N/2-point transformation architecture is similar to the N-point transformation architecture. It also consists of a butterfly circuit, core multiplication module, and add/sub circuit. This means that 64-point transformation has a 32-point transformation circuit, which in turn has a 16-point transformation circuit to the 4-point transformation circuit which finally has a 2-point transformation embedded within each other. However, in order to respect the throughput some N-point DCT/iDCT-2 circuits will be duplicated. If the transform size is larger than 32 (N >=32) the input vector carries one column/row of the size N. Otherwise, the input vector carries 32/N columns/rows of size N. This means that the N-point DCT/iDCT-2 circuit will be duplicated 32/N times when N<32. For example, if 16-point DCT/iDCT-2 is selected, the input vector carries two column/rows (depending on the direction) of 16 samples. As a result, the 16-point DCT/iDCT-2 is instantiated twice. Table 7.5 shows the number of duplications per transform size for the throughput of 32 samples/cycles. Table

Table 7.5 number of circuits per transform size

Input Size	Number of duplication
4-point	8
8-point	4
16-point	2
32-point	1
64-point	1

7.6 summarizes the cost of each transform size in term of Total Number of Multiplexers (TNM) and number of Constant Multiplierss (CMs) and the Total Number of Adders and Subtracts (TNAS) used for butterfly and add/sub modules. Table 7.6 shows the cost of each N-point shared design, these

Table 7.6 Cost summary

Transform size	Operators	TNM	CM	TNAS
4-point	{64,36,83}	24	8	12
8-point	{18, 50, 75, 89} + 4-point operators	56	16+8 = 24	16+12 = 28
16-point	{9, 25, 43, 57, 70, 80, 87, 90} + 8-point operators	120	64+24 = 88	32+28 = 60
32-point	{4, 13, 22, 31, 38, 46, 54, 61, 67, 73, 78, 82, 85, 88, 90, 90} + 16-point operators	248	256+88 = 344	64+60 = 124
64-point	{91, 90, 90, 90, 88, 87, 86, 84, 83, 81, 79, 77, 73, 71, 69, 65, 62, 59, 56, 52, 48, 44, 41, 37, 33, 28, 24, 20, 15, 11, 7, 2} + 32-point operators	504	1024+344 = 1368	128+124 = 252
Global cost (all sizes)	64-point operators	504	1536*	416**

* = 8*8+16*4+64*2+256+1024
** = 12*8+16*4+32*2+64+128

circuits are duplicated depending on the transform size (N) as shown in Table 7.5. As a result, the cost of each transform size is multiplied by the corresponding duplications number to support the 32 samples/cycle throughput. The mutualization between the DCT-2 and its inverse consumes half the number of multiplications compared to the cost of a solution with the two transforms implemented separately. However, the mutualization adds several multiplexers at the input and output interfaces which may impacts the overall surface of the circuit.

7.4 Results and comparison

Each of these designs have its own challenges. For instance, the LFNST/DCT design uses Regular Multiplier (RM) while the DCT-2/iDCT-2 design uses CM. As a result, instead of comparing both designs in term of number of multipliers, we present the main characteristics of each design as shown

in Table 7.7.

Table 7.7 DCT/iDCT vs DCT/LFNST

	Shared DCT/LFNST	Shared DCT/iDCT
Throughput (samples/cycle)	32	32
Number of modules supported	3: LFNST, iLFNST and fDCT-2	2: fDCT-2 and iDCT-2
Multiplications	RM	CM
Number of RM/CM (all sizes)	1024	1536
Number of RM/CM (64-point is eliminated)	512	512
Multiplexers (all sizes)	-	504
Adders/subtractors (all sizes)	-	416

In Table 7.7 we present the different characteristics of each shared design. As we can notice, the main advantage of the LFNST/DCT design is that it supports more modules than the DCT/iDCT-2 design (3 vs 2). On the other hand, the performance of this design is measured based only on the DCT-2 consumption. The LFNST has no effect on the total number of multiplications used. In fact, it consumes much less than the DCT-2. On the other hand, the shared DCT/iDCT design is balanced, both modules are impacting the performance of the design. Moreover, taking advantage of the butterfly decomposition the unified DCT/iDCT-2 design uses the same core computation modules for both transforms, at the expense of minor input/output multiplexers.

7.5 Conclusion

In this chapter we have proposed a study of a hardware design for two shared designs between the forward DCT-2 and the LFNST, and between the forward DCT-2 and its inverse. For both solutions we presented their costs is term of multiplications, supported modules, additions and other input/output constraint at a speed of 32 samples per cycle. This study has shown that for the two shared modules, the overall resource consumption and the cost in terms of the number of multipliers is measured entirely on the consumption of DCT-2 core. The LFNST on the other hand consumes much less than the DCT-2. As a result, it is more interesting to adopt a shared design between the DCT-2 and its inverse while keeping the LFNST as an independent core. In term of hardware area consumption it is more convenient to adopt this solution. Finally, for the real time VVC encoder the DCT-2/iDCT-2 shared design was adopted and integrated as an efficient strategy used to lower the cost of the transform chains.

Part III

Lightweight hardware implementation of VVC Adaptive Loop Filter for ASIC platform

Chapter 8

Background and state of the art of the VVC ALF

8.1 Introduction

The Adaptive Loop Filter (ALF) was firstly considered as a tool candidate for High Efficiency Video Coding (HEVC), but finally it was not included in the standard [74, 75]. However, it was then standardized as one of the in-loop filters in Versatile Video Coding (VVC) [3] standard. In-loop filters are located in the decoding loop of the encoder and aim to enhance the perceived quality of the decoded video sequence by eliminating the blocking, ringing or blurring artifacts generated by previous decoding stages. The VVC in-loop filter block relies on three main filters: Deblocking Filter (DF) [41], Sample Adaptive Offset (SAO) [42] and ALF. The encoder estimates the optimal filter parameters that maximize, the most, the objective quality of a block. These parameters are then transmitted to the decoder so that the in-loop filters of the decoder can use them to optimally filter the reconstructed frame. The role of the three filters are defined as follows. The first one, DF, aims to remove the artifacts that appear in the edges of the Coding Unit (CU). It will be mainly in charge of applying a smoothing filter to edges in order to remove blocking artifacts. A different type of smoothing filter can be applied to a block according to the properties of its neighboring blocks. The strength of the filter coefficients will be determined by the specific values of the edge pixels and the Quantization Parameter (QP). The SAO comes second with the objective to reduce the undesirable visible artifacts such as ringing. The SAO filter classifies pixels of a Coding Tree Unit (CTU) into two different categories. If the reconstructed samples in a specific category have a smaller value than the original ones, a positive offset is added to reduce the existing error. On the other hand, if samples in a specific category have higher values than their corresponding original ones, a negative offset will be applied. The final block of the in-loop filters in VVC is the ALF. The main role of the ALF is to reduce visible artifacts such as ringing and blurring by reducing the mean absolute error between the original image and the reconstructed one. ALF is the main purpose of this chapter. Figure 8.1 illustrates the VVC in-loop filters carried out at the decoder. The in-loop block is fed by a reconstructed picture, which is then processed by the three filters in the order illustrated in Figure 8.1: DF followed by the SAO followed by the ALF. The ALF filter delivers the reconstructed and filtered picture for the display at

the decoder side and for the Decoded Picture Buffer (DPB) and for temporal prediction at the encoder side. It can also be applied at the post-processing stage of any decoded video to enhance its visual quality before the display.

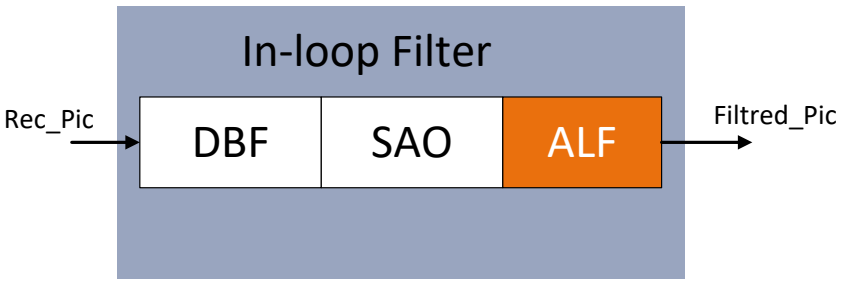


Fig. 8.1 VVC In loop filters

In this chapter the background of the ALF is described, and it is organized as follows. Section 8.2 presents the different filters used by the ALF for the Luma and Chroma components. Section 8.3 introduces the existing hardware implementations of the ALF. Finally, Section 8.4 concludes the chapter.

8.2 ALF Filters

ALF uses Wiener filters for its core operation. Wiener filters are designed to minimize the Mean Squared Error (MSE) between the original samples and the filtered samples of a signal. The original samples refer to the original frame from the video sequence and the filtered samples to the reconstructed picture after the ALF (ALF's output). The ALF is divided into several processing steps as shown in Figure 8.2. The first one performs block partitioning, where the CTU is divided into multiple elementary macro-blocks of size 4×4. These macro-blocks are then filtered using the same set of coefficients whether for Luma or Chroma CTBs. The filter's coefficients selection is detailed separately in the next section. ALF uses different filters for Luma and Chroma components. For the Luma, it uses a 7×7 Diamond Shape Filter (DSF). The filter coefficients of the 7×7 DSF can be fixed or signaled in the Adaptation Parameter Set (APS). For the Chroma components, the ALF uses two filters. The first one is a 5×5 DSF filter and, unlike the 7×7 DSF, the filter coefficients can only be signaled in the APS. The second filter is the Cross-Component Adaptive Loop Filter (CCALF) diamond filter. This latter uses the co-located Luma samples to filter the Chroma ones. Like the 5×5 DSF, the coefficients are only signaled in the APS. Figure 8.3 shows the ALF filters as defined in VVC specification.

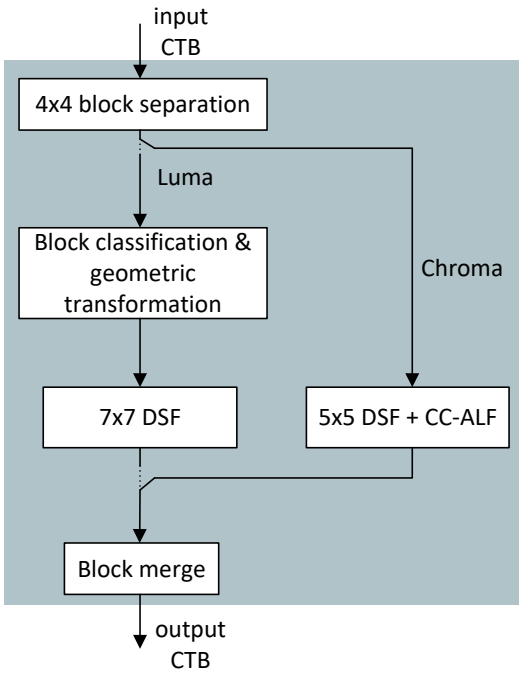


Fig. 8.2 ALF filtering steps for Luma and Chroma Coding Tree Block (CTB)s

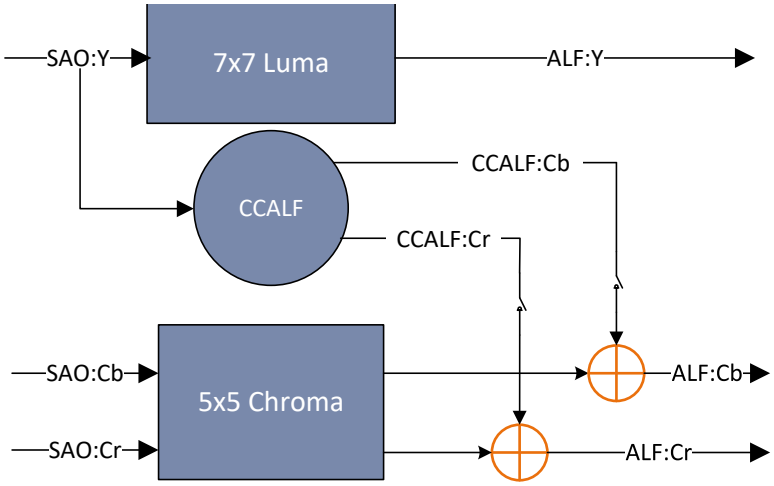


Fig. 8.3 ALF Luma and Chroma filters in the VVC decoder.

The remaining of this section will be divided in three parts. The first one addresses the Luma filter and its coefficient derivation, the second one investigates the Chroma filtering while the last one addresses the Virtual Boundary (VB) concept.

8.2.1 Luma filter

8.2.1.1 Block classification

The ALF classification process is computed only for the Luma component. Each 4×4 Luma macro-block is categorized into one out of 25 classes. The classification index C is derived based on the macro-block directionality D and a quantized value of local pixels activity A computed as follows:

$$C = 5D + \widetilde{A} \tag{8.1}$$

To calculate D and $\widetilde{A}$, the sum of horizontal g_h, vertical g_v and two diagonal g_{d1}, g_{d1} gradients are first computed using a 1-D Laplacian as follows:

$$g_h = \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} H_{k,l}, \quad \text{with } H_{k,l} = |R_{k-1,l} - 2R_{k,l} + R_{k+1,l}| \tag{8.2}$$

$$g_v = \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} V_{k,l}, \quad \text{with } V_{k,l} = |R_{k,l-1} - 2R_{k,l} + R_{k,l+1}| \tag{8.3}$$

$$g_{d1} = \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} D1_{k,l}, \quad \text{with } D1_{k,l} = |R_{k-1,l-1} - 2R_{k,l} + R_{k+1,l+1}|. \tag{8.4}$$

$$g_{d2} = \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} D2_{k,l}, \quad \text{with } D2_{k,l} = |R_{k-1,l+1} - 2R_{k,l} + R_{k+1,l-1}|, \tag{8.5}$$

where indices (i, j) refer to the coordinates of the upper left sample within the 4×4 macro-block and $R_{i,j}$ indicates a reconstructed sample at coordinate (i, j). *maxY* and *minY* are derived based on the macro-block position relative to the virtual boundary. The maximum values that can take *minY* and *maxY* are 2 and 5, respectively.

To reduce the complexity of the block classification, the sub-sampled 1-D Laplacian calculation is applied. As shown in Figure 8.4, the same sub-sampled positions are used for gradient calculation in all directions. This means that Eq (8.3), (8.2), (8.4) and (8.5) are only applied when both k and l are even numbers or both are odd numbers. This reduces the computation complexity by half. The blue pixels in Figure 8.4 refer to the current 4×4 macro-block to be filtered. All samples of this macro-block shares the same class value C.

After computing the sum of gradient in all directions, the maximum and minimum values of the macro-block between the horizontal and vertical directions, and between the two diagonal directions,

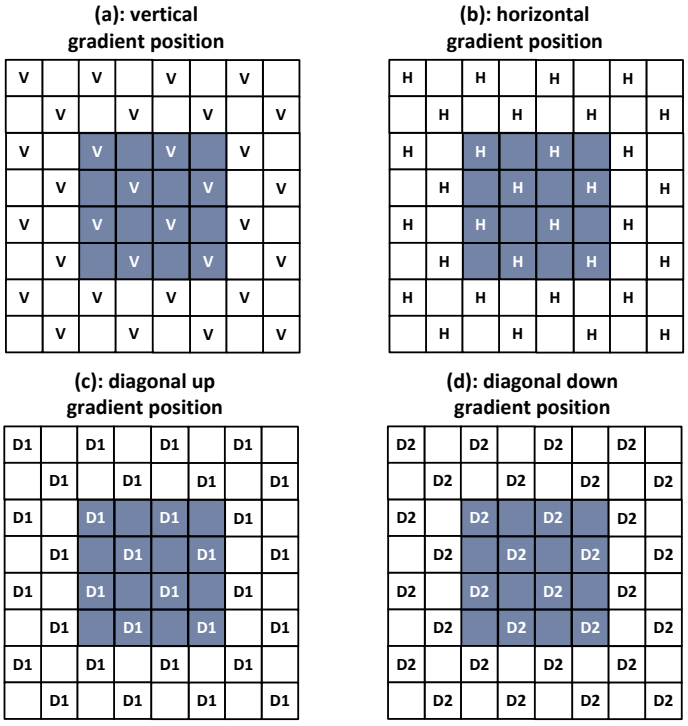


Fig. 8.4 Sub-sampled gradients in vertical, horizontal and two diagonal directions. Empty samples are not used for gradients computation. The samples of current 4×4 macro-block are highlighted in blue color.

are computed by Eq (8.6)

$$\begin{aligned} g_{hv}^{max} &= max(g_h, g_v), \\ g_{hv}^{min} &= min(g_h, g_v), \\ g_d^{max} &= max(g_{d0}, g_{d1}), \\ g_d^{min} &= min(g_{d0}, g_{d1}). \end{aligned} \tag{8.6}$$

According to its direction, each 4×4 macro-block will be assigned an integer value D in the interval $[0, 4]$. The value D will be then used to select the macro-block class. To determine the value of the directionality D, the g_{hv}^{max} and g_d^{max} are compared against each other with two thresholds t_1 and t_2 as follows:

- **Step 1**: if both $g_{hv}^{max} \leq t_1\, g_{hv}^{min}$ and $g_d^{max} \leq t_2\, g_d^{min}$ are true, then D is set to 0.
- **Step 2**: if $g_{hv}^{max}/g_{hv}^{min} > g_d^{max}/g_d^{min}$ then continue from step 3, otherwise continue from step 4.
- **Step 3**: if $g_{hv}^{max} > t_2\, g_{hv}^{min}$, D is set to 2, otherwise D is set to 1.
- **Step 4**: if $g_d^{max} > t_2\, g_d^{min}$, D is set to 4, otherwise D is set to 3.

This process is summarized as a pseudo code in Algorithm 1.

Algorithm 1: Pseudo-code to compute D

Data: g_{hv}^{max} maximum between horizontal and vertical gradients
g_{hv}^{min} minimum between horizontal and vertical gradients
g_{d}^{max} maximum between first and second diagonal gradients
g_{d}^{min} minimum between first and second diagonal gradients

```
if g_hv^max ≤ t1 g_hv^min and g_d^max ≤ t2 g_d^min then
    D ← 0;
else
    if g_hv^max / g_hv^min > g_d^max / g_d^min then
        if g_hv^max > t2 g_hv^min then
            D ← 2;
        else
            D ← 1;
    else
        if g_d^max > t2 g_d^min then
            D ← 4;
        else
            D ← 3;
```

Result: D

The quantized activity value $\widetilde{A}$ is calculated by Eq (8.7):

$$\widetilde{A} = \left\lfloor \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} (V_{k,l} + H_{k,l}) \right\rceil_0^4, \tag{8.7}$$

where i and j are the positions within the image relative to the top-left corner , $\lfloor A \rceil_0^4$ stands for a quantized value of A to the nearest integer and saturated in the interval $[0,4]$. Finally, based on the values of D and $\widetilde{A}$, the corresponding class value C is derived by Eq (8.1). The class value is used to determine the filter coefficients to filter the Luma component.

Before applying the filter, ALF performs geometric transformations to the filter kernel. This is equivalent to applying these transformations to the samples in the filter support region. The idea is to make different filtered macro-blocks more similar by aligning their directionality.

8.2.1.2 Geometric transformations

Three geometric transformations are introduced including rotation, vertical flip and diagonal flip which are expressed by Eq (8.8), (8.9) and (8.10), respectively.

$$f_r(k,l) = f(N-l-1,k), \qquad c_r(k,l) = c(N-l-1,k), \tag{8.8}$$

$$f_v(k,l) = f(k,N-l-1), \qquad c_v(k,l) = c(k,N-l-1), \tag{8.9}$$

$$f_d(k,l) = f(l,k), \quad c_d(k,l) = c(l,k), \tag{8.10}$$

where N is the size of the filter and $0 \leq k,l \leq N-1$ are coefficients coordinates, such that coordinates $(0,0)$ refer to the upper left corner and coordinates $(N-1,N-1)$ to the lower right corner. The transformations are applied to the filter coefficients $f(k,l)$ and to the clipping values $c(k,l)$ depending on gradient values calculated for the current macro-block. In fact, the $c(k,l)$ clipping values are values that bound the results of the filtering operation and they can be signaled in the APS or can be determined based on the bit-depth of the frame. The relation between the transformation and the four gradients of the four directions are summarized in Table 8.1.

Table 8.1 Mapping between the gradient values and the geometric transformations

Gradient value comparison	Transformation type
$g_{d2} < g_{d1}$ and $g_h < g_v$	No transformation
$g_{d2} < g_{d1}$ and $g_v < g_h$	Diagonal flip
$g_{d1} < g_{d2}$ and $g_h < g_v$	Vertical flip
$g_{d1} < g_{d2}$ and $g_v < g_h$	Rotation

8.2.1.3 Luma 7×7 diamond shape filter

The ALF uses a 7×7 2D DSF for the Luma component as shown in Figure 8.5:(a). The DSF is a point-symmetrical filter which allows factoring the coefficients two by two and thus reducing the number of multiplications by half. Since multipliers need much larger chip area than an adder or a subtractor, this factorization reduces significantly the processing part area by eliminating half of the multiplications.

Thanks to the symmetry feature, the total number of coefficients for the Luma filter is 12. These coefficients are of two types: 1) the first one is the fixed filter coefficients, 2) the second one is the customized filter coefficients for Luma signaled in APS. VVC defines the fixed filter coefficients using two matrices. The first one, named *AlfFixFiltCoeff*, contains the 64 fixed Luma filters defined by VVC. The second matrix, named *AlfClassToFiltMap*, holds 16 sets of 25 indices that can be used in the *AlfFixFiltCoeff* matrix of size 64×12 to determine the Luma filter. The first index of the *AlfClassToFiltMap* is signaled in the slice header and can be in the range 0 to 15. The second index is determined based on the classification value C of the current $4\times$ 4 macro-block, where C is an integer bounded in the interval $[0,24]$. Based on these indices, the Luma filter index is retrieved from the *AlfClassToFiltMap* thus finding the 12 filter coefficients stored in the *AlfFixFiltCoeff*.

The second type of luma coefficients is the "Luma coefficients signaled in the APS". This alternative uses coefficients that are signaled to the decoder and retrieved from the APS. In one APS, up to 25 sets of Luma filter coefficients and clipping values could be signalled. The choice of these coefficients is determined by the encoder. This process depends on a certain criteria based on the pixel level analysis and rate-distortion optimisation. Each 4×4 macro-block uses one filter set which can be selected among 25 filters using the classification value of the macro-block.

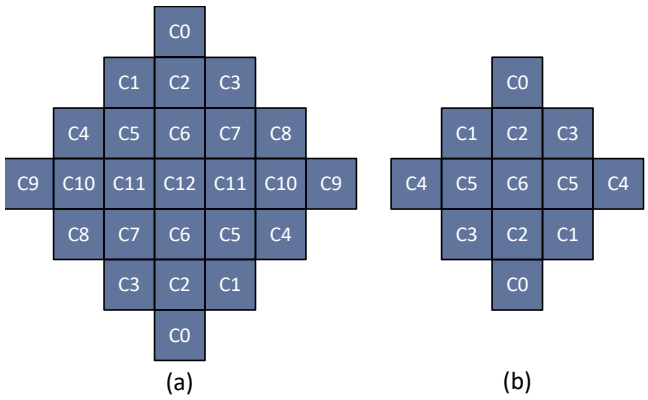


Fig. 8.5 Diamond Shape filters for Luma and Chroma, (a): Luma 2D 7×7, (b) Chroma 2D 5×5

8.2.2 Chroma filters

The ALF uses two DSF for the Chroma components, the first one is a 5×5 DSF shown in Figure 8.5:(b) and the second one, shown in Figure 8.6, is a cross-component filter which uses co-located Luma samples to filter Chroma ones. Like the Luma filter, the 5×5 DSF is point symmetrical where only 6 of coefficients are signaled in the APS. On the opposite, the CCALF filter does not have the symmetrical propriety and needs a total of 7 coefficients for its filtering process. Unlike the Luma filter, the coefficients of Chroma filters can only be signaled in the APS, therefore there is no classification for Chroma macro-blocks nor fixed coefficients. Eight sets of coefficients are transmitted in the APS, while the index of the used set is signaled at the CTB level.

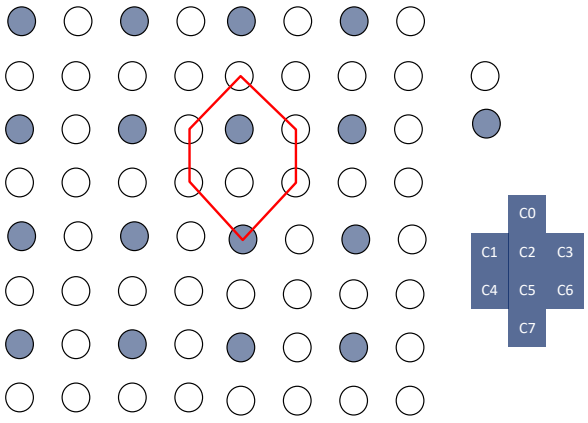


Fig. 8.6 Cross component 2D Diamond Shape filter, 420 sub-sampling example. Blue circles represent the Chroma component while white circles are the Luma component.

8.2.3 Filtering process

At decoder side, when ALF is enabled for a CTB, each $R(i,j)$ sample within the block is filtered, resulting in sample value $R'(i,j)$ as shown in Eq (8.11).

$$R'(i,j) = R(i,j) + (((\sum_{k\neq 0}\sum_{l\neq 0} f(k,l) \times clip\_fun(R(i+k,j+l) - R(i,j), c(k,l))) + 64)/2^7) \quad (8.11)$$

where $f(k,l)$ denotes the filter coefficients, $clip\_fun$ the clipping function and $c(k,l)$ denotes the decoded clipping parameters. The indexes k and l vary between $-L/2$ and $L/2$where L denotes the filter length. The clipping function $clip\_fun = min(y, max(-y, x))$.

8.2.4 Virtual Boundary Filtering Process

To reduce the line buffer requirement of ALF, modified block classification and filtering are employed for the samples near horizontal CTU boundaries. For this purpose, a virtual boundary is defined as a line by shifting the horizontal CTU boundary with "N" samples, with N equal to 4 for the Luma component and 2 for the Chroma components. Modified block classification is applied for the Luma

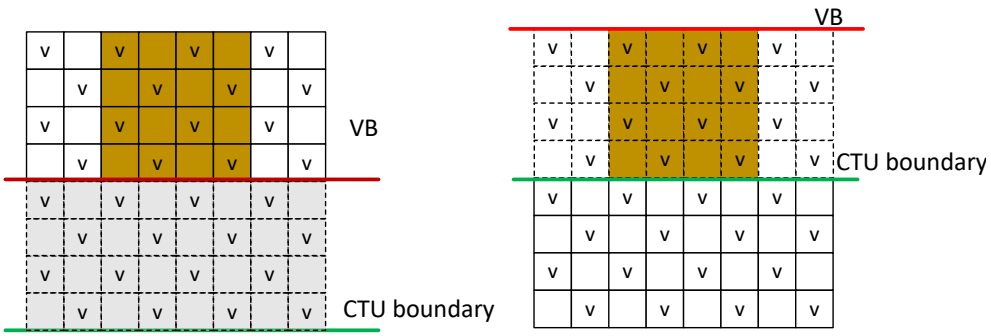


Fig. 8.7 Modified block classification at virtual boundaries

component as depicted in Figure 8.7. For the 1D Laplacian gradient calculation of the 4×4 block above the VB, only the samples above the virtual boundary are used. Similarly, for the 1D Laplacian gradient calculation of the 4×4 block below the virtual boundary, only the samples below the virtual boundary are used. The quantization of activity value A is accordingly scaled by considering the reduced number of samples used in 1D Laplacian gradient calculation. For filtering processing, symmetric padding operation at the virtual boundaries are used for both Luma and Chroma components. As shown in Figure 8.8, when the sample being filtered is located below the virtual boundary, the neighboring samples that are located above the virtual boundary are padded. Meanwhile, the corresponding samples at the other sides are also padded, symmetrically. The same padding method is applied for the 5×5 DSF at VB.

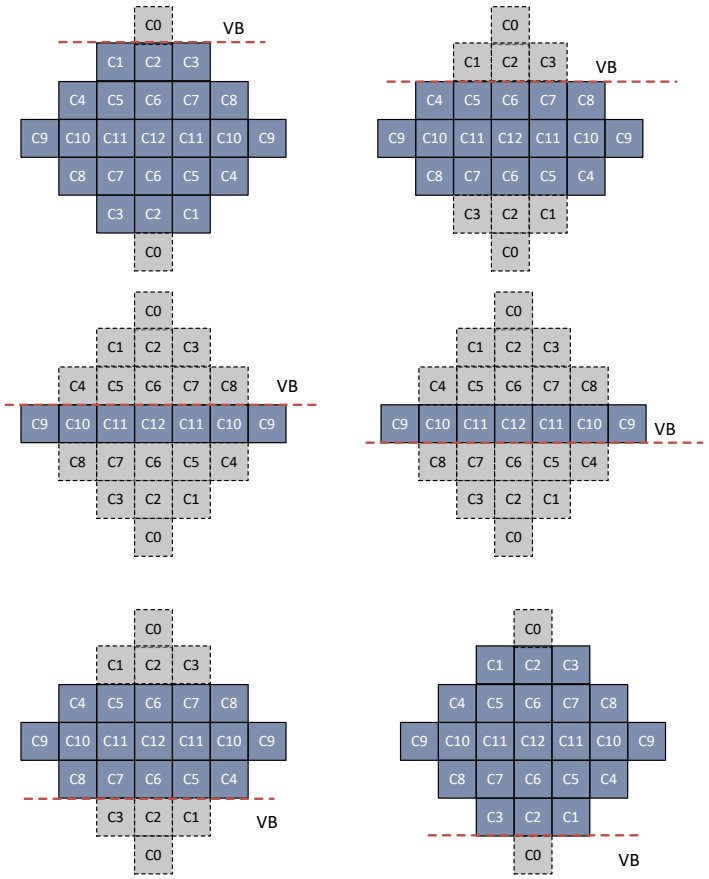


Fig. 8.8 Modified ALF filtering for Luma component at virtual boundaries

8.3 Related Work

In this section, a brief description of existing hardware implementations of the ALF is given. Several ALF hardware implementations have been proposed in the literature, since it was previously proposed for the HEVC standard. Du *et al.* [76] proposed a hardware architecture for both deblocking filter and ALF. They implemented the ALF based on macro-block size in raster scan order. The ALF needs the eight surrounding 4×4 final-deblocked blocks to process a 4×4 block. The authors concluded that the deblocking and ALF orders require large local buffers to enable data reuse for deblocking. Ruhan *et al.* [77] proposed a shared hardware architecture for the 5×5, 7×7 and 9×9 diamond shaped ALF filters using the fact that large filter sizes have small ones nested in them. The paper takes advantage of the ALF symmetrical characteristics to reduce the total number of multiplications from 13 to 7 for the 5×5 DSF, from 25 to 13 for the 7×7 DSF and from 39 to 20 in the case of 9×9 DSF. This is realized by applying addition operations to the samples that have the same corresponding coefficient. The work in [77] was synthesized targeting an Field Programmable Gate Array (FPGA) platforms of 40 nm technology. Fabiane *et al.* [78] proposed efficient hardware designs for the ALF cores of sizes 5×5, 7×7 and 9×9. Similar to [77], the symmetrical feature of the ALF is explored to reduce

the total number of multipliers. The three architectures were synthesized targeting FPGA platform of 90 nm technology. Results showed that the proposed designs were able to process 110, 102 and 98 frames per second in 1080p resolution for the 5×5, 7×7 and 9×9 filter sizes, respectively. In [79], the same author presented a hardware architecture for the 5×5 ALF core targeting the same family of FPGA platforms. Xin et al. in [80] proposes a hardware design targeting ASIC platforms that reduces 60% of the memory access, saves approximately 50% of the hardware resources including adders and multipliers cost, increases the throughput and makes the hardware configurable for luma and chroma components. The synthesis result demonstrates that the architecture achieves a throughput of 4k@120fps and a maximum frequency of 367MHz. Xiang et al. in [81] proposes an algorithm to reduce the complexity of the adaptive loop filter. This optimization was for intra coded frames only. In Joint Exploration Model (JEM) v3.0 the ALF was applied to the entire I frame; the paper proposes to apply ALF only to inhomogeneous region in the frame. The algorithm computes the gradient to decide were the region is homogeneous or not.

However, the aforementioned studies do not address the gradient computation nor the memory management of the ALF which are considered as the most critical parts of the design. Also, a fixed system latency for all filter sizes is an important feature to ensure efficient interaction of the ALF module with other decoder modules. Moreover, the ALF decision module is not studied which is a essential part at the encoder side.

8.4 Conclusion

In this chapter we presented the different filters used by the ALF and the related hardware designs studies. The ALF is part of the in-loop carried-out at the end of the coding loop. It aims at reducing visible artifacts from the reconstructed image by performing adaptive filtering to minimize the MSE between original and reconstructed samples based on Wiener filtering. For hardware design the ALF comes with many challenges: filter implementation, gradient computation, memory storage, cross-components filtering and the decision process for the encoder side. Therefore, the hardware ALF module should be carefully designed leveraging all optimisations to reach the required latency and throughput while minimising the memory and hardware area.

Chapter 9

ALF HW design for 4K VVC decoder

9.1 Introduction

In this work, as in the iMultiple Transform Selection (MTS) and iLow Frequency Non-Separable Transform (LFNST) works presented in Chapter 6, the aim is to design a real time Versatile Video Coding (VVC) video decoding chip that supports 4kp60 video resolution with a target frequency of 450Mhz. The chip is expected to decode the target resolution over two Application-Specific Integrated Circuit (ASIC) cores. As a result, each core is expected to process 4Kp30 resolution in real time. In practice, the required throughput for a specific tool of the VVC decoder depends on the video resolution specified in time and pixel domains, which are measured with respect to the fps and the pixel area, respectively. Sub-sampling of Chroma components can additionally affect the throughput performance. Since the Adaptive Loop Filter (ALF) module is expected to processes pixels in $N \times N$ units, it is convenient to use their number per second to specify and measure the throughput. In fact, the ALF is applied at the Coding Tree Unit (CTU) level which is the basic compression unit in VVC. The CTU size is flexible and can be specified in the Sequence Parameter Set (SPS) as 128×128, 64×64, 32×32, or 16×16 in units of Luma samples. Table 9.1 shows the detailed performance for

Table 9.1 ALF throughput performance per CTU size for 4Kp30 with 4:2:2 resolution @450Mhz

Luma CTB size	Chroma CTB size	unit size (pixels)	Throughput (unit/s)	Clock budget (cycles/unit)	ALF (pixels/cycle)
128×128	128×64	32768	15188	16384	2
64×64	64×32	8192	60750	4096	2
32×32	32×16	2048	243000	1024	2
16×16	16×8	512	972000	256	2

each CTU size supported by the ALF. It highlights the cycle budget available for each block and the required number of processed pixels per cycle with respect to the target frequency. As it can be noticed, in order to support 4Kp30 resolution, the ALF is required to process 2 pixels every clock cycle.

In addition, to facilitate chaining between CTUs, it should sustain a fixed system latency regardless of the CTU size, which corresponds to CTU of size 128×128. The module should be able to compute the following:

- For Luma: Gradient, classification and 7×7 Diamond Shape Filter (DSF)
- For Chroma: 5×5 DSF and 3×4 Cross-Component Adaptive Loop Filter (CCALF)

The ALF processor control interface is summarized in Table 9.2. A positive pulse in *alf_enable* launches the ALF process, with the CTU size and block position defined by *ctu_size* and *alf_blk_pos*. The ALF coefficients information is defined in the interface with prefix *Luma_..* and *Chroma_...* When *alf_fix_luma_en* is enabled, fixed Luma coefficients are used for the 7×7 DSF. Otherwise, Luma in Adaptation Parameter Set (APS) coefficients are utilized. Every eight clock cycles, the input 4×4 block will be refreshed with new data through out the *alf_data_in* interface. The position of the input block is computed according to the raster scan order. The position counters are defined in top level design for each color component, separately. Therefore, the input data is synchronized with its position at the input of the module. For the Chroma component, only coefficients in the APS are used. This latter is signaled throughout the *Chroma_coeff_aps* interface. The cross-component filters are applied only when *alf_cc_cr_en* or *alf_cc_cb_en* are enabled for their corresponding color components. The ALF one starts generating the results after a fixed system latency and then, thanks to a pipelined architecture, it continuously generates filtered blocks every eight clock cycles without any stall.

This Chapter presents a novel hardware-architecture for the VVC ALF filter, considering its three filters: 7×7, 5×5 and 3×4 CCALF. This design is based on the last working draft of the VVC standard [6]. The proposed design sustains a fixed throughput of 2 pixels/cycle with a fixed system latency that enables the processing of 4K video at 60 frames per second in 4:2:2 Chroma sub-sampling. The proposed design is integrated into a professional video decoder chip supporting multiple video coding standards. The following sections are organized as follows. Section 9.2 describes the top level design of the ALF module. Section 9.3 describes the different input scanning along with the most optimal scanning order. Section 9.4 describes the memory configurations of the proposed input scanning order. The proposed design for the gradient module and for the core ALF filters are described in Section 9.5. Section 9.6 presents the experimental setup and the synthesis results. Finally, Section 9.7 concludes the chapter.

9.2 Top level design

Figure 9.1 shows the top level design of the ALF module. The proposed design is decomposed of three main parts: the memory management, the gradients computation and the filtering module.

The memory module (MEMORY_ACCESS_MODULE in Figure 9.1) is the most critical part of the ALF hardware design, and our solution aims to reduce the cost of the memory. To find the lowest memory cost, we investigated multiple input scanning orders for Luma and Chroma components. The optimal solution for memory was found when the scanning between Luma and Chroma samples

Table 9.2 ALF design interfaces.

Signals	I/O	#BITS	Description
clk	I	1	System Clock
rst_n	I	1	System reset, active low
alf_enable	I	1	Activation pulse to start
alf_cc_cr_en	I	1	Cross component enable for Cr
alf_cc_cb_en	I	1	Cross component enable for Cb
alf_fix_luma_en	I	1	Enable fixed coefficient for Luma
alf_color_comp	I	2	00:Y, 01:Cr and 10:Cb
alf_data_in	I	16×10	Data input interface, 16 samples, 10-bits samples
alf_blk_pos	I	9	4x4 block position within the CTU (in raster scan)
ctu_size	I	2	00: CTU16, 01: CTU32, 10: CTU64, 11: CTU128
pic_height	I	11	Picture height
pic_width	I	11	Picture width
Luma_coeff_aps	I	12×8	Luma coefficient in APS, 25 set of 12 coefficients
Chroma_coeff_aps	I	6×8	Chroma coefficient in APS, 8 set of 6 coefficients
Chroma_cc_alf_coeff_cr_aps	I	7×8	Cross component coefficient in APS for Cr
Chroma_cc_alf_coeff_cb_aps	I	7×8	Cross component coefficient in APS for Cb
alf_data_out	O	16×10	Output bus of 16 samples
alf_out_enable	I	1	End of 4x4 block filtering

is interlaced. The proposed design uses two memories: the first stores Luma and Chroma macro-blocks whereas the second one stores only Luma macro-blocks. The memory handler takes care of memory accesses and allows the selection of the appropriate group of macro-blocks required to apply the filtering process. The memory management and dimension are investigated in the next sections. The gradient module is applied only when Luma component is selected. It receives the blocks from the memory, computes the sum of gradients and determines the classification value used to select the appropriate Luma filter coefficients. Finally, the filtering module aims to take advantage of the fact that larger filters have smaller ones nested inside. This allows the reuse of hardware resources used for small filters to implement larger ones. Thus, it is possible to reduce the total hardware cost in comparison with an architecture that implements the three filters independently. For ASIC devices, this reduction in terms of resource consumption is important, since these devices are limited in terms of both size and battery.

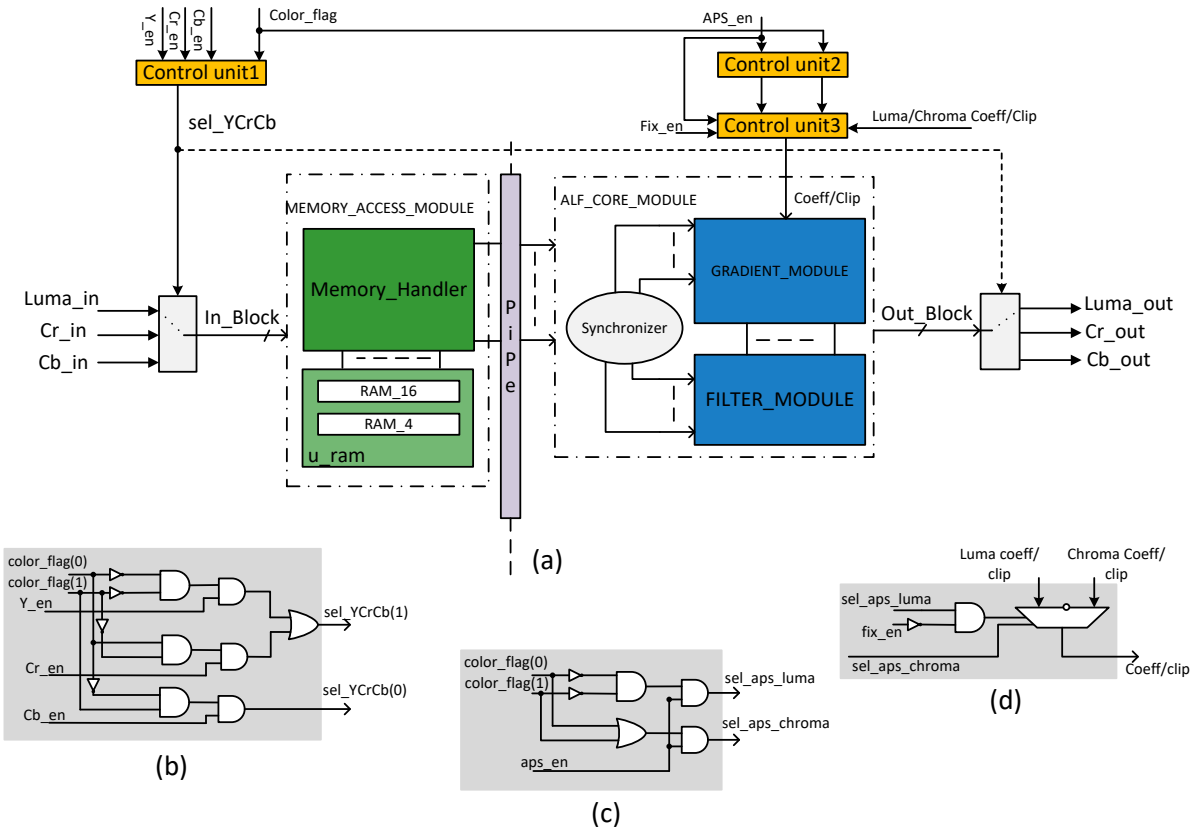


Fig. 9.1 Proposed hardware ALF architecture, (a): Unified design for ALL filters, (b): *Control unit 1:* this unit will initiate the start signals depending on the input color component. (c): *Control unit 2:* this unit will initiate the coefficients in APS enable signals depending on the input color component, (d): *Control unit 3:* depending on the APS enable and the Fix_en signals, this unit will propagate the appropriate coefficients and clip values to the core module.

9.3 Core module requirements and input scanning

The ALF is applied at the CTU level. CTUs within a picture are scanned in raster order from left to right and from top to bottom. To reduce the memory usage and eliminate the use of line buffer (storing an entire frame line of Luma and Chroma macro-blocks), the ALF introduces a set of clipping methods applied for different samples close to the boundaries. However, this does not eliminate totally the need for a memory, it only reduces the total number of macro-blocks to store.

9.3.1 Required surrounding samples

For the Luma component, the 7×7 DSF is applied on every 4×4 macro-block using one among 25 filters determined by the classification results. To compute the class and apply the filter, each Luma macro-block requires its eight surrounding neighbours: top left (TL), top (T), top right (TR), left (L), right (R), bottom left (BL), bottom (B) and bottom right (BR), as shown in Figure 9.2:(a). Within the

Luma Coding Tree Block (CTB), macro-blocks are raster scanned and when the BR block is received, the current block, in orange, is processed. Unlike Luma, the Chroma components has two filters: the 5×5 DSF and the CCALF filters, the coefficients of these filters are determined by the encoder and signaled, via the APS, to the decoder. The 5×5 DSF is applied to macro-blocks of size of 4×2 for 4:2:0 chroma sub-sampling and to macro-blocks of size of 4×4 for 4:2:2 chroma sub-sampling. Same as for the Luma component, to filter one Chroma macro block, the eight surrounding macro-blocks(TL, T, TR, L, R, BL, B and BR) are required, as shown in Figure 9.2: (b) and (c). Following the raster scan order, when BR block is received, the current block, in orange, is filtered. On the other hand, the CCALF filter uses co-located Luma pixels to apply 4×3 CCALF filter. As shown Figure 9.2:(b) for 4:2:0 Chroma sub-sampling example, a group of surrounding Luma samples are required to apply the CCALF filter. As we can notice, the number of required samples for 4:2:0 and for 4:2:2 are slightly different. The 4:2:2 sub-sampling uses two bottom neighboring lines to filter bottom samples while for the other case it uses only one bottom line. Finally, the filtered Chroma block is the resulting sum of the two filters (CCALF + 5$\times$ 5 DSF).

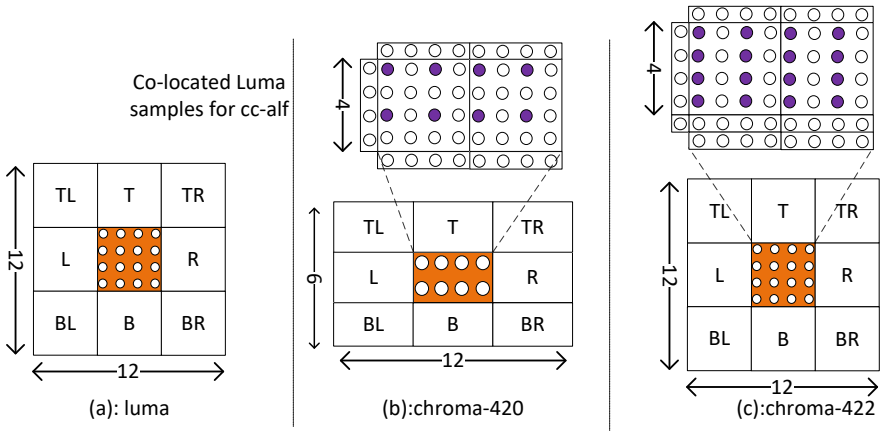


Fig. 9.2 (a) required macro blocks for Luma component (b) required macro blocks for Chroma component with 4:2:0 sub-sampling (c) required macro blocks for Chroma component with 4:2:2 sub-sampling

9.3.2 Input scanning

In order to reduce the memory cost, multiple input scanning orders were studied. These scanning scenarios are expected to support 4:2:0 and 4:2:2 Chroma sub-sampling. It is important to not that in order to filter a Chroma macro-block, Luma macro-blocks needs to be already stored in the memory. In addition, at the input of the ALF it is assumed that the input samples can be Luma or Chroma at any time and that the scanning order within the Luma or Chroma CTB is a free axis. Based on that, three main scanning scenarios were defined:

(A): ***Filter the Luma CTB first***

In this scenario, the entire Luma CTB is filtered at first then the Chroma components (Cr and Cb) are processed in second place. Due to the cross-component filtering for Chroma, it is essential to store the entire Luma CTB before moving to the Chroma filtering. Figure 9.3 shows the memories required the 4:2:2 Chroma sub-sampling. Different scanning order for macro-blocks can be used for this scenario, however, regardless of scanning order the same memory size will be used. Colored blocks in Figure

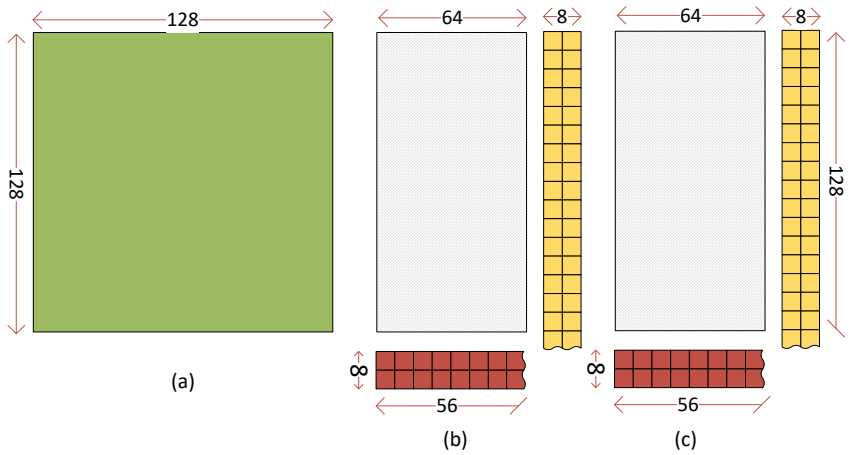


Fig. 9.3 Scenario (A) required memory for 4:2:2 sub-sampling example, (a) Luma blocks, (b) Chroma Cr blocks, (c) Chroma Cb blocks

9.3 indicates the blocks that stored in the memory. The red and yellow blocks refers to the column and line buffers, respectively, for both Chroma components. The Luma CTB is colored in green, which means that it is entirely stored in the memory for this case scenario. For 4:2:0 Chroma sub-sampling, the same line and column buffers are used, however, with half the size since the Chroma blocks are reduced by half. Table 9.3 shows the overall memory cost for 4:2:0 and 4:2:2 Chroma sub-sampling. As we can notice from Table 9.3, the total memory size required for this scenario is 193.28 Kbits for

Table 9.3 Scenario (A) memory cost for 10-bits samples.

	4:2:2		4:2:0	
	Samples	Kbits	Samples	Kbits
Luma	(128×128)	163.84	(128×128)	163.84
Chroma lines	$2\,(56 \times 8)$	8.96	$2\,(56 \times 4)$	4.48
Chroma columns	$2\,(128 \times 8)$	20.48	$2\,(64 \times 8)$	10.24
Total	19328	193.28	17856	178.56

4:2:2 and 178.56 Kbits for 4:2:0 Chroma sub-sampling.

(B):*Filter the Chroma CTBs first*

In this scenario, the entire Chroma CTBs are processed before starting Luma CTB. Due to the cross-component filtering, the Chroma blocks can not be finished until the Luma blocks are received. As a result, both filtered Chroma CTBs are stored in the memory. These blocks are released only when the cross-component finish the filtering. Figure 9.4 shows the memories required the 4:2:2 Chroma sub-sampling example. Same as the first scenario, this one is also independent from the scanning order. Whatever the scanning is used for the input macro-blcoks, the same memory is used.

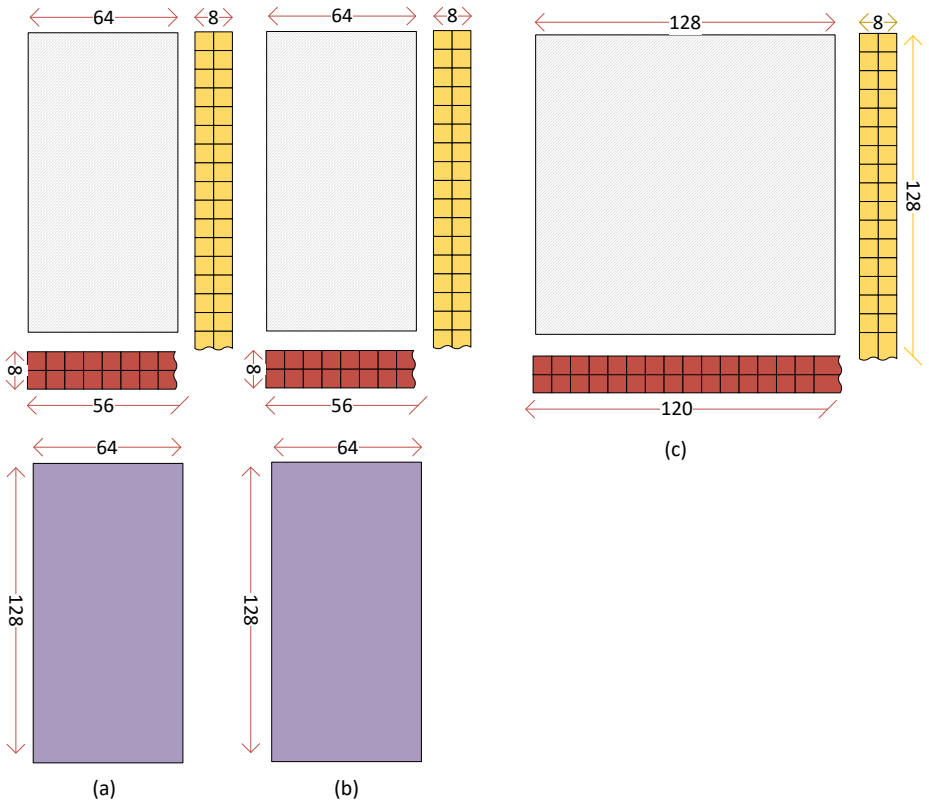


Fig. 9.4 Scenario (B) required memory for 4:2:2 sub-sampling example, (a) and (b) show the Chroma Cr and Cb filtered and unfiltered blocks for memory, (c) Luma memory blocks

For 4:2:0 Chroma sub-sampling, the same buffers are used for the Chroma components but with half the size. Table 9.3 shows the overall memory cost for 4:2:0 and 4:2:2 Chroma sub-sampling. As we can notice from Table 9.4, the total memory size required for this scenario is 213.12 Kbits for 4:2:2 and 116.38 Kbits for 4:2:0 Chroma sub-sampling.

(C): *Alternation between Luma and Chroma macro-blocks*

In this scenario, the aim is to mix the input Luma and Chroma macro-blocks with the purpose of finding the best alternation that provides the lowest memory. The idea is to eliminate the need to store the entire Luma or Chroma CTBs for the CCALF filtering. Figure 9.5 shows the memory cost

Table 9.4 Scenario (B) memory cost for 10-bits samples.

	4:2:2		4:2:0	
	Samples	Kbits	Samples	Kbits
Luma lines	120×8	9.6	120×8	9.6
Luma columns	128×8	10.24	128×8	10.24
Chroma lines	$2\,(56\times 8)$	8.96	$2\,(56\times 4)$	4.48
Chroma columns	$2\,(128\times 8)$	20.48	$2\,(64\times 8)$	10.24
Filtered Chroma	$2\,(128\times 64)$	163.84	$2\,(64\times 64)$	81.92
Total	21312	213.12	11638	116.38

required the 4:2:2 Chroma sub-sampling example. Unlike the previous scenarios, the scanning order is critical for this scenario which will be presented in the next sub-section. The proposed scanning order reduces heavily the memory cost compared to other scenarios.

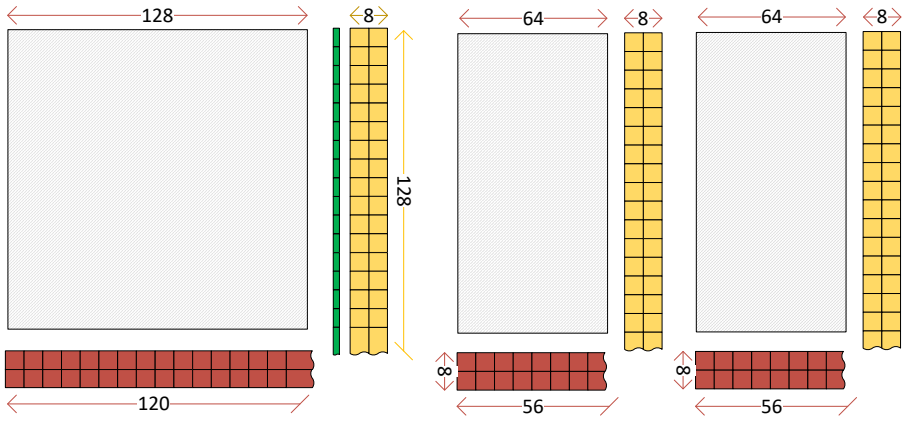


Fig. 9.5 Scenario (C) required memory for 4:2:2 sub-sampling example: (a) Luma Line, Column and CCALF storage, (b) Chroma Cr Line and Column storage, (c) Chroma Cr Line and Column storage

For 4:2:0 Chroma sub-sampling, the same buffers are used for the Chroma components but with half the size. One particular difference between the two Chroma sub-sampling is noticed for Chroma blocks that are close to the virtual boundary. One line of 4×2 macro-blocks is not filtered using the cross-component filter for 4:2:2 sub-sampling. This latter does not impact the overall memory size, however, it impacts the total number of memory accesses. Table 9.5 shows the memory cost for 4:2:0 and 4:2:2 Chroma sub-sampling.

Table 9.6 shows the memory usage for each scanning order scenario for 4:2:0 and 4:2:2 chroma sub-sampling with 10-bits samples. For scenarios (A) and (B), the raster scan and the z-scan are almost similar in terms of memory usage, therefore, Table 9.6 reports the results for raster scan. As it can be noticed, the proposed scanning order reduces significantly the memory usage compared to the other two scenarios. For scenario A, the entire Luma CTB is stored so that the CCALF can be applied when starting the Chroma components. Compared to the proposed scanning order, it

Table 9.5 Scenario (C) memory cost for 10-bits samples.

	4:2:2		4:2:0	
	Samples	Kbits	Samples	Kbits
Luma lines	120×8	9.6	120×8	9.6
Luma columns	128×8	10.24	128×8	10.24
Chroma lines	$2\,(56\times 8)$	8.96	$2\,(56\times 4)$	4.48
Chroma columns	$2\,(128\times 8)$	20.48	$2\,(64\times 8)$	10.24
CCALF column	(128×1)	1.28	(128×1)	1.28
Total	5056	50.56	3584	35.84

consumes around four times more memory for both 4:2:0 and 4:2:2 subs-ampling. For scenario B, the entire Chroma CTBs is processed using the 5×5 DSF, however it is not possible to start the CCALF since the Luma components are still not available. As a result, the filtered Chroma CTBs are stored in memory, waiting for the CCALF filter results, which is applied only when the Luma samples are available. Compared to scenario C (the proposed scanning order), it consumes more than two times memory for 4:2:0 and more than three times for 4:2:2 sub-sampling. It can be noticed that scenario (C) consumes much less memory compared to the other two scenarios. For both Chroma sub-sampling, it consumes around 25% of the maximum required memory depending on Chroma sub-sampling. With this it can be concluded that the lowest memory size can be achieved efficiently if Luma and Chroma macro-blocks are interlaced. Therefore, a good alternation between Luma and Chroma blocks is the best solution for finding the lowest memory cost. This led to the definition of a new scanning order which is an alternation between color components of the same CTU line. The ALF module alternates between Luma and Chroma macro blocks by reading for each CTU line one $4\times 2/4\times 4$ Chroma Cr macro-block followed by one $4\times 2/4\times 4$ Chroma Cb macro-block followed by two 4×4 Luma macro-blocks. The proposed scanning order is described in details in the next subsection.

Table 9.6 Memory usage (Kbits) comparison for different scanning order scenarios.

	4:2:2			4:2:0		
Scenarios	A	B	C	A	B	C
7×7 DSF	163.84	19.84	19.84	163.84	19.84	19.84
5×5 DSF	29.44	193.28	29.44	14.72	96.64	14.72
3×4 CCALF	0⋆	0⋆	1.28	0⋆	0⋆	1.28
Total size	193.28 (90%)	213.12 (100%)	50.56 (26%)	178.56 (100%)	116.48 (65%)	35.84 (20%)

⋆ Required samples are already in memory.

9.3.3 Proposed scanning order

As explained, every 4×4 Luma macro-block needs its eight surroundings neighbors: from TL to BR macro-block. The current macro-block can start processing once the BR is received. This implies that

the top two macro-block lines of the Luma CTB need to be stored in memory. These lines of 4×4 macro-blocks are measured on the maximum size of the supported CTB for VVC standard which is 128×128, which results in storing 8×120 pixels. Using these two macro-block lines (8 samples wide) in a rotating memory, the entire Luma CTB can be filtered. Moreover, the last column of a CTB can be filtered once the next CTB is available. As a result, macro-blocks belonging to the last column of the CTB needs to be stored in memory along with its TL, L, BL, T and B neighbors. Consequently, for Luma components, two lines and two columns of 4×4 macro blocks measured for 128×128 CTBs are stored. Thus, the total Random-Access Memory (RAM) size for Luma components is 20.48 Kbits for 10-bits pixels.

Like Luma, every 4×4 (or 4×2 for 4:2:0) Chroma macro-block is filtered, using the 5×5 DSF, by its eight surrounding neighbors, as shown in Figure 9.2:(b) and (c). This implies the storage of two Chroma macro-block lines in the memory. These macro-block lines are measured on 128×64 CTBs for 4:2:2 sub-sampling and on 64×64 for 4:2:0 sub-sampling, which results in a total of 4×64 pixels. Same as the Luma component, the last column of the Chroma CTB can only be filtered once the first column of the next CTB is available. Therefore, two additional columns are stored in memory. As a result, two lines and two columns of size 4×4 (4×2 for 4:2:0) are stored. This latter corresponds to a total of 4×56 pixels for line memory and 8×128 (8×64 for 4:2:0) pixels for column memory. Thus, the total RAM size for the 5×5 DSF for both Chroma components (Cr and Cb) is 29.44 Kbits for 10-bits pixels for 4:2:2 Chroma sub-sampling and 14.72 Kbits for 4:2:0 Chroma sub-sampling.

- *Scanning order for 4:2:0 sub-sampling:*

  Figure 9.6 shows an example of Luma/Chroma scanning order for a CTU of size 32×32 for 4:2:0 chroma sub-sampling. The Luma, Cb and Cr chroma samples are illustrated in Figure 9.6 by gray, blue and yellow colors, respectively. Figure 9.6:(b) shows the alternation pattern between Luma and Chroma macro blocks. One 4×2 Chroma Cr followed by one 4×2 Chroma Cb followed by two 4×4 Luma macro-blocks are scanned for every CTU line. The macro-blocks of consecutive numbers and color in the figure belong to the same CTB. For each color component, four CTBs are shown with different color intensities. As it can be noticed, the ALF is actually applied to a shifted version of the CTB with the same size, delimited by the green rectangle in Figure 9.6:(a) with respect to the virtual boundary positions. Once macro-block of position 31 for Cr and Cb components are available, the macro-block of positions 26 from the previous line can be processed. On the other hand, for Luma component the macro blocks of positions 53 and 54 are filtered once macro-blocks of positions 62 and 63 are received, respectively. For all color components, the distance between received macro-block and the current macro block is one CTB line, which can be seen in the input/output sequencing in Figure 9.6:(b).

- *Scanning order for 4:2:2 sub-sampling:*

  Figure 9.7 shows an example of Luma/Chroma scanning order for a CTU of size 32×32 for 4:2:2 chroma sub-sampling. Same as the previous Figure, the Luma, Cb and Cr samples are

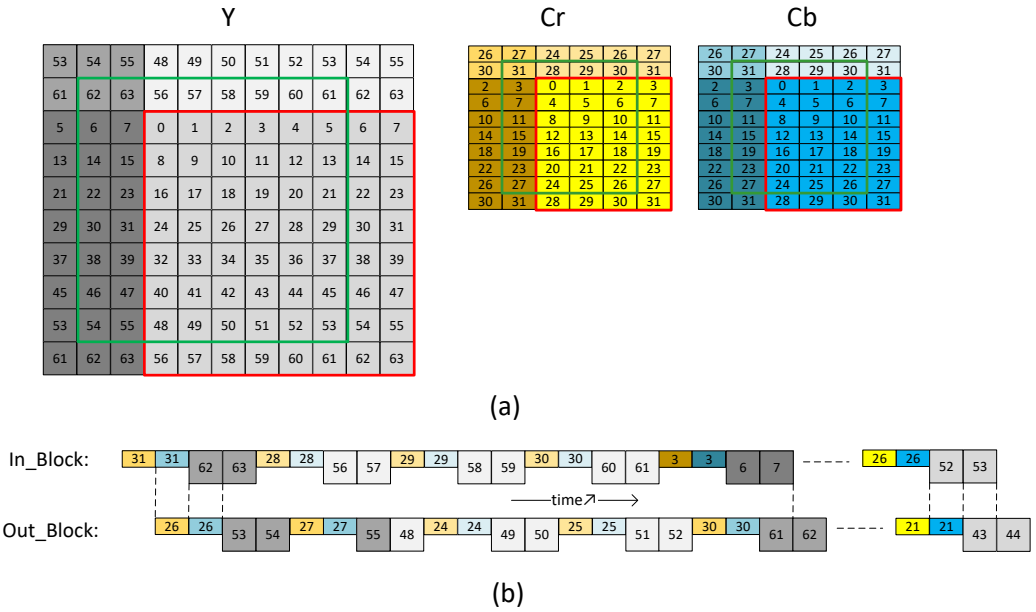


Fig. 9.6 ALF macro block scanning for 4:2:0 sub-sampling. (a): CTU32 example: the red rectangle is the current CTB. The green rectangle represents the block processed by the ALF (bounded by the VB). (b) input/output macro-blocks sequencing: one 4×2 Cr macro-block followed by one 4×2 Cb macro-block followed by two 4×4 Y macro blocks.

illustrated in Figure 9.7 by gray, blue and yellow colors, respectively. Figure 9.7:(b) shows the alternation pattern between Luma and Chroma macro blocks. Unlike the 4:2:0 sub-sampling, this one alternates between Luma and Chroma blocks of size 4×4. It reads one 4×4 Chroma Cr followed by one 4×4 Chroma Cb followed by two 4×4 blocks. However, one exception is located for blocks near the Virtual Boundry (VB) (shopped blocks in Figure 9.7:(a)). In this case, only 4×2 Chroma blocks are processed instead of 4×4. This is due to the fact that the samples directly below the VB are not filtered using the cross-component filter. As reported in studies [82–85], two Chroma lines are eliminated for CCALF because of the lack of their co-located Luma samples at the time the Chroma CTB is processed. Same as for the 4:2:0 case, for all color components, the distance between the received macro-block and the current macro-block is one CTB line, which can be seen in the input/output sequencing in Figure 9.7:(b). The Chroma (Cr or Cb) macro-block of position 30 can start processing once the macro-block of position 35 is received. The block of position 35 is shopped in the Figure because the CCALF filter is not supported for this position, however, it supports the 5×5 DSF. On the other hand, for Luma component, the macro-blocks of positions 53 and 54 are filtered once macro-blocks of positions 62 and 63 are available, respectively.

As it can be noticed from the timelines of Figure 9.6:(b) and 9.7:(b), the CCALF filter can be applied in parallel with the 5×5 DSF taking advantage of the fact that co-located Luma macro blocks are already retrieved from the memory when applying the 7×7 DSF. As a result, no further memory accesses or memory storage are required for this filter. However, since the last column of the Chroma

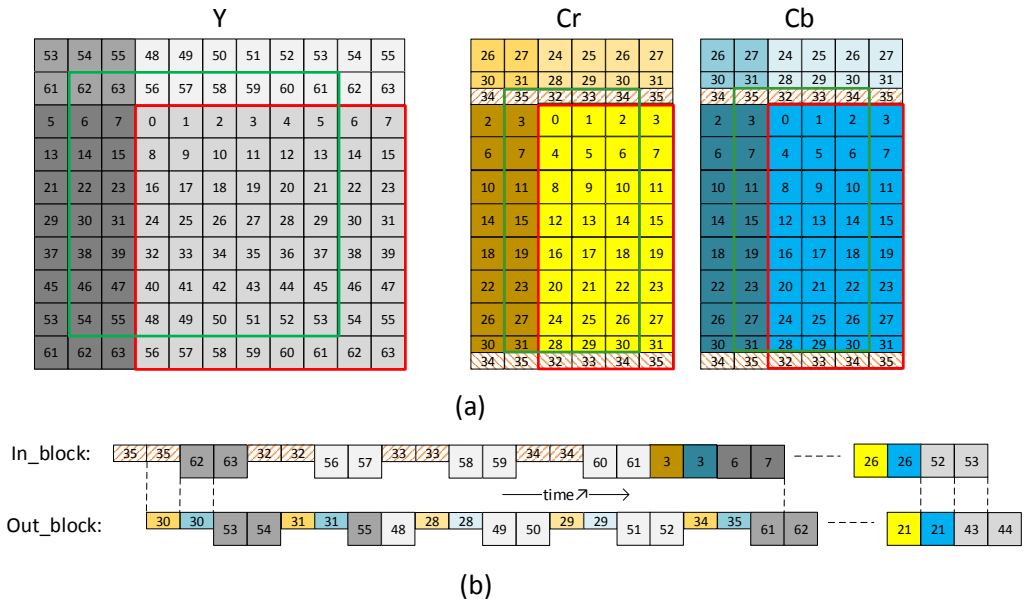


Fig. 9.7 ALF macro-block scanning for 4:2:2 sub-sampling. (a): CTU32 example: the red rectangle is the current CTB. The green rectangle represents the block processed by the ALF (bounded by the VB). (b) input/output macro-blocks sequencing: one 4×4 Cr macro-block followed by one 4×4 Cb macro-block followed by two 4×4 Y macro blocks.

CTB is filtered at the reception of the first column of next CTB, a column of 1 sample wide is stored in memory. Without that, when the second CTB is available, the CCALF filter would be missing the left co-located samples. Thus, the memory required for CCALF is 1.28 Kbits (128×1 for 10-bits samples). As a result, for the two Chroma filters, a total of 30.72 Kbits (29.44 for 5×5 DSF + 1.28 for CCALF) of RAM are stored for 4:2:2 including a total of 16 Kbits (14.72 for 5×5 DSF + 1.28 for CCALF) for 4:2:0 Chroma sub-sampling. Finally, the total memory used for both Luma and Chroma components is 50.56 Kbits of RAM. Using this amount of RAM, both Chroma sub-sampling can be supported. Compared to other scanning scenarios, the proposed scanning order reduces the memory by almost 75%.

9.4 ALF Random Access Memories

As explained in the previous sections, the proposed scanning order allows for a total of 50.56Kbits of RAM. In fact, these samples are stored over two different memories as shown in Figure 9.8. The first one, *RAM_16*, stores Luma and Chroma macro-blocks of size 4×4 with an interface of size 160-bits (10-bits samples) which allows the access of a 4×4 macro-block every clock cycle. Using this memory we are able to compute the gradients, the 7×7 and 5×5 DSF filters. The second one, *RAM_4*, stores samples used for the CCALF filter. This memory stores an entire column of 128 samples. Its interface is equal to four samples which means a 40-bits size interface. Therefore, a 1×4 macro-block is accessed every clock cycle.

To sustain a solid pipeline with a throughput of 2 samples/cycle, memory accesses should be

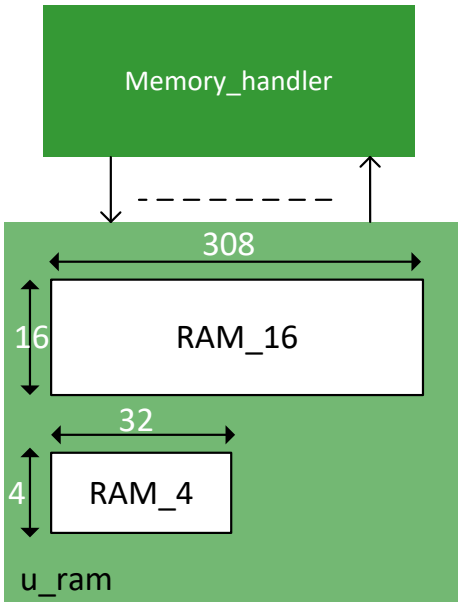


Fig. 9.8 ALF Random Access Memories

carefully scheduled so that all reading and writing operations required for a single macro-block are performed in a window of 8 clock cycles. This time frame is actually bounded by the throughput of the design. In fact, to process a 4×4 macro-block (Luma or Chroma), an eight clock cycle window is the maximum available time frame which is computed as follows: 16 samples / 2 (samples/cycle) = 8 cycles. However, as shown in Figure 9.2, every macro-block (Luma or Chroma) needs all its eight surrounding neighbors for the filtering operations which requires at least 9 clock cycles or 9 memory operations: 8 readings + one writing. Therefore, it is essential to find a method that reduces these operations in order to sustain the predefined rate. In this part, we present a solution that decreases significantly the memory operations by exploiting all possible advantages between ALF processing stages.

For both Chroma and Luma components, the total number of required memory accesses are performed over 9 clock cycles which is larger than the constraint of 8 cycles. To reduce this, shared neighboring macro-blocks are used to eliminate some reading operations. In fact, every two successive marco-blocks have at least six macro-blocks in common, as shown in Figure 9.9. Macro-block at position "N" can make use of six neighbor macro-blocks that are already retrieved when processing the previous macro-block of position "N-1". The chopped region depicts the shared blocks between the two consecutive operations. Using this, the number of memory operations for macro-block “N” are less than the one for position “N-1”. In fact, position “N” needs only three memory operations which are two readings and one writing. As a result, the clock cycles required for two consecutive macro-blocks are as follows: 9 clock cycles for the first macro-block and 3 clock cycles for the second one. Thus, a total of 12 clock cycles for 2 consecutive macro-blocks, while we have a window of 16 clock cycles (each block has a window of 8 cycles). Therefore, by taking advantage of the shared region between every two consecutive operations, we are able to sustain a throughput of 2

samples/cycle for the Luma and Chroma filters.

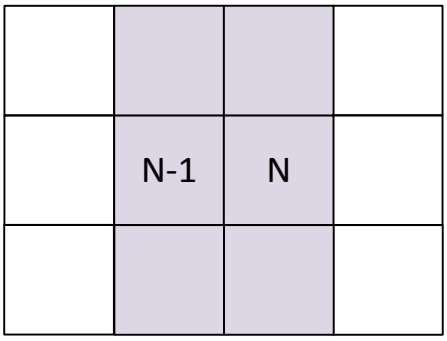


Fig. 9.9 Shared region between two successive macro-blocks at the same line

9.5 Luma/Chroma unified core architecture

After selecting the right group of macro blocks from the memory, the second stage of the ALF depicted by the ALF_CORE_MODULE in Figure 9.1: (a) is executed. This module is composed of two sub-modules: the gradient module and the unified core filtering one. The gradient module, applied only when Luma samples are processed, computes the sum of gradients and determines the classification value used to select the appropriate Luma filter coefficients. The filtering module is a unified design that uses 26 multipliers to apply the three Luma/Chroma filters: 7×7 DSF, 5×5 DSF and 3×4 CCALF.

9.5.1 Gradient Module

The gradient module is actually formed by three sub-modules: the gradient sum, the Read-Only Memory (ROM), and the kernel transpose. Figure 9.10 depicts the top level design of the gradient module. The first one is the gradient sum module in which the sum of gradients is computed following Eq (8.3), (8.2), (8.4) and (8.5) presented in Chapter 8 of the ALF background. Using the gradient result, the classification value is determined and the filter coefficients to be used for the Luma component are selected from the ROM. When one of the Chroma components is selected, the gradient module is bypassed and the Chroma coefficients are delivered directly to the filtering module. These coefficients (whether Luma or Chroma) are synchronized with the required data and delivered to the filtering module. The second module is the ROM which is used to store the fixed Luma coefficients defined by the VVC standard. It stores a total of 64 sets of Luma filters where each set has 12 coefficients. Therefore, the total ROM size is 6144-bits ($64 \times 12 \times 8$-bits). Luma components have also coefficients that are signaled in the APS. In all cases, from the ROM or from the APS, one Luma filter is selected using the filter index and the classification value. However, before transmitting these coefficients to the filtering module a certain transposition is applied to the filter kernel. This transposition is performed by the transpose module, where, depending on the gradient sum results a rotation, vertical flip or horizontal flip is applied to the Luma filter, according to the conditions presented in

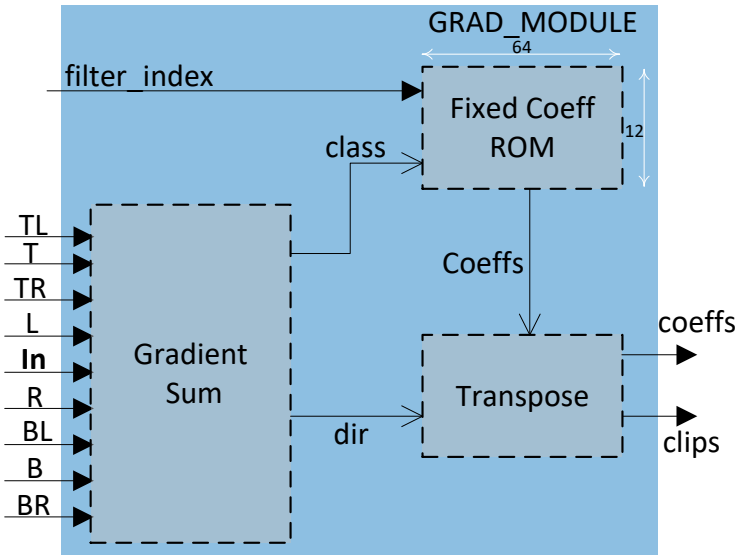


Fig. 9.10 Gradient sum module

Table 8.1 in Chapter 8 of the ALF background.

9.5.2 Core filters

The last sub-module of the ALF core is the filtering module, shown in Figure 9.11. At the input, the module receives nine 4×4 macro blocks containing the current macro block with its eight surroundings, along with the filter clipping and coefficients values. At the output, it delivers the filtered macro block at a throughput of 2 pixels/cycles. It is composed of 3 pipeline stages including S1: pixel selection, S2: clip/sum and S3: mult_stage. The S1 stage is used for pixel selection, depending on the filter to be applied, it selects the required pixels for a given (x [0:3], y [0:3]) position. The selected pixels are then transmitted, via the "pix_array" register to the pre-multiplication stage (S2). This latter performs both clipping and additions. As known, ALF filters are symmetrical, this allows to add symmetrically positioned pixels before applying the multiplication. The results of this stage are delivered to the third and final stage via two separate arrays (pix1_array, pix2_array), each representing one pixel. In order to process 2 pixels/cycle, this stage uses a total of 26 Regular Multiplier (RM). Finally, the third stage delivers the 4×4 ALF filtered macro block.

9.6 Experimental and Synthesis Results

The proposed design was implemented using the VHSIC Hardware Description Language (VHDL). A state-of-the-art logic simulator [66] is used to test the functionality of the ALF memory and core multiplication modules. The test strategy is as follows: a software implementation of the ALF has been developed, based on the specification of the last version of the VVC standard. Using self-check techniques, the bit accurate test-bench compares the simulation results with those obtained using the reference software implementation. In the next sections, the coding gain and the software complexity

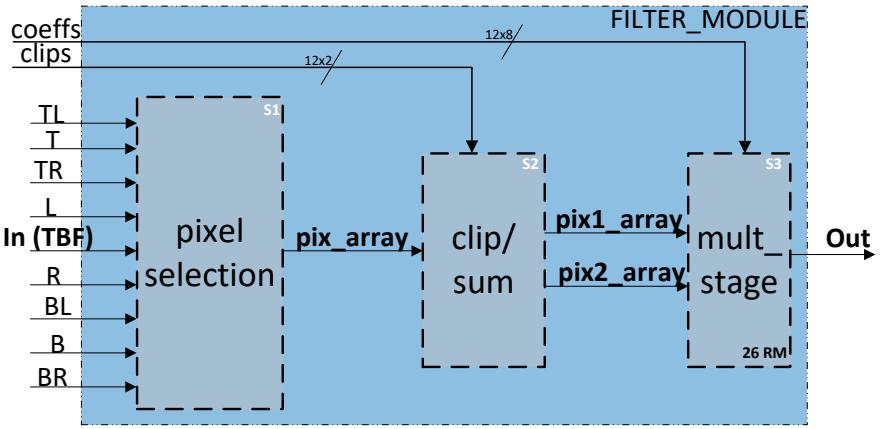


Fig. 9.11 ALF filter module

of the ALF are presented along with the synthesis results.

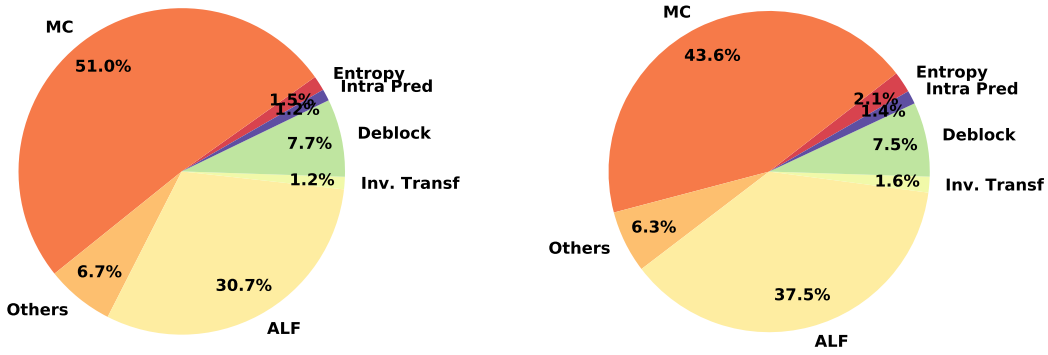
9.6.1 ALF coding gain and software complexity analysis

The coding gain and software complexity of the ALF and CCALF modules under the VVC common Test conditions in three main coding configurations including all intra (AI), random access (RA) and low delay B are analysed [69]. Table 9.7 gives the coding losses and the complexity reductions at both encoder and decoder when the ALF, CCALF and Sample Adaptive Offset (SAO) tools are turned off.

Table 9.7 Performance (%) in terms of Bjøntegaard Delta Rate (BD-BR) and run time complexity when ALF, CCALF and SAO tools turned off in VTM10.0 [5]. Evaluations performed under the VVC common test conditions [6].

Disabled tool	All Intra Main 10					Random Access Main 10					Low Delay B Main 10				
	BD-BR			Complexity		BD-BR			Complexity		BD-BR			Complexity	
	Y	U	V	*EncT*	*DecT*	Y	U	V	*EncT*	*DecT*	Y	U	V	*EncT*	*DecT*
ALF	2.20	12.81	12.43	100	92	4.34	19.31	19.06	97	89	4.12	25.06	19.58	99	92
CCALF	−0.14	9.69	8.55	100	97	−0.13	13.88	13.73	100	99	−0.15	18.58	13.79	100	97
SAO	0.01	0.15	0.20	100	91	0.08	0.14	0.31	97	96	0.11	0.43	1.23	97	93

It can be noticed that disabling ALF and CCALF tools introduces a high coding losses for Chroma components for all configuration with a maximum of 25.06% and 18.58% BD-BR (U component) for Low Delay configuration when these two tools are disabled, respectively. Compared to SAO, disabling the ALF tools introduces larger coding loss specifically for the Chroma components. This is due to the fact that the ALF uses two filters for the Chroma components: the 5×5 DSF and the CCALF. Figure 9.12, shows the complexity of the ALF compared to other VVC tools in terms of decoding time. As it can be noticed, the ALF, embedding the CCALF, is considered as one of the most complex tool of the VVC standard. Compared to the Deblocking Filter (DF), ALF takes more than 24% of decoding complexity for and it comes second after the Motion Compensation (MC) for both Quantization Parameter (QP) values.



(a) QP 37 (b) QP 32

Fig. 9.12 Complexity distribution of decoding tools in the VTM software for two different QPs.

9.6.2 Synthesis Results

Table 9.8 gives the results of ALF module in terms of chip surface and memory consumption targeting ASIC platforms at 450 Mhz. This table also shows how resources are distributed over the sub-modules of the ALF. The ALF design operates at 450 MHz with a total of 66997 um^2 and a total 50.56 Kbits of RAM used to store two lines of 4×4 macro-blocks for each color component and one additional Luma column of 1 sample wide measured for CTUs of size 128×128. The memory access module consumes 48% of design total resources while the gradient and the filter modules consume only 23% and 29%, respectively. The proposed design was integrated into a real time video decoder enabling to decode a 4k video at 60 frame per second (fps) for 4:2:2 and 4:2:0 Chroma sub-sampling. The decoder also supports the three recent MPEG video standards including Advanced Video Coding (AVC)/H.264, High Efficiency Video Coding (HEVC)/H.265 and VVC/H.266 standard.

Table 9.8 Synthesis results for ALF @2 p/c at 450 MHz on ASIC 28-nm platform for typical condition of 85 Celsius with 0.8V input voltage

	Memory access module	Gradient module	Filter module
Number of mult.★	0	0	26
Combinational area	9368	10398	12910
Non combinational area	20281	4368	5115
Total area (um^2)	30758	14767	18025
RAM (bits)	50560	-	-

★ The number of multipliers is included in the total number of gates.

To further analyse the performance, the ALF design is synthesised for two input voltages at a

typical condition of a temperature of 85 Celsius, the result can be seen in Table 9.9. As it can be noticed from the table, for 0.8V the power consumption is about 9 mW which is almost 30% larger than the one with the 0.65V which is about 6 mW. Reducing the power consumption comes at the expense of larger chip area, the one targeting 0.65 V is 12% larger than the former one. In Table 9.10,

Table 9.9 Power results for ALF @2 p/c at 450 MHz on ASIC 28-nm platform for typical condition of 85 Celsius

	0.8 V	0.65 V
Power (mW)	9.5819	6.6921
Total area (um^2)	66997	76521

the result of this study are compared to our previous study [67] done for the VVC inverse transform block. Both studies were done in the same environment and got integrated in a real time 4Kp60 multi-standard decoder, which allows an accurate comparison. Both the ALF and transform block can be reduced to a matrix multiplication problem. Taking advantage of the butterfly decomposition and the zeroing for large block sizes, the inverse MTS can be computed using only 32 multipliers for two pixels/cycle. The inverse LFNST on the other hand, is applied to the top left 8×8 corner of the input transform block, this allows it to use only 32 multipliers for the same throughput. The largest filter for the ALF is the 7 × 7 DSF and by taking advantage of the symmetrical feature of the filter, the ALF is able to process 2 pixels/cycle using only 26 multipliers. Compared to the transform blocks, the ALF uses less multipliers which allows it to consume 11% less area than the LFNST and 24% less area than the MTS. Since the ALF requires neighboring samples to apply the filters, it uses about 50Kbits of memory, on the other hand, the 1-D MTS and the LFNST does not require any memory where the blocks are independently processed.

Table 9.10 Adaptive Loop Filter vs Inverse transform blocks (MTS + LFNST) @2 p/c

	ALF	iMTS	iLFNST
number of mult.★	26	32	32
Total area	66997	89082	74592
RAM (bits)	47360	-	-

★ The number of multipliers is included in the total number of gates.

A fair comparison with the state of the art solutions is quite difficult since most of works focus on earlier versions of the ALF and do not support the memory management and the gradient computation. Table 9.11 gives the key performance features of state-of-the-art Field Programmable Gate Array (FPGA) and ASIC-based works. The only work that discusses the memory is found in [76], however it focuses on the chaining between the DF and ALF and does not provide the results of memory usage. Compared to previous works, the net memory results related to the ALF and CCALF are provided, with a study of a shared architecture for all ALF filters based the last version of the VVC standard [86].

The solutions proposed in [77–79] discuss only the calculation part of ALF filters and do not

address the memory. Unlike these solutions, our solution provides a complete study for the ALF hardware implementation and discusses the most challenging parts which are the memory and gradient modules. Like the work of [77, 78], a shared architecture is proposed for multiple filters taking advantage of the fact that smaller filters are nested within larger ones by using a number of regular multipliers. Compared to [77], our solution consumes less multipliers while supporting the same throughput. This is because [77] supports a previous version of the ALF that uses 9×9 DSF which was not included in the VVC specification. Compared to [78], our solution supports larger throughput and more Chroma sub-sampling, this comes at the use of more multipliers. Solution in [79] provides a study only for the 5×5 DSF which is considered to be incomplete. To the best of our knowledge, the proposed solution is the first one to address a complete study for ALF based on the last version of the VVC standard [86].

The proposed design was integrated into a video decoder that supports three different video standards including AVC/H.264, HEVC/H.265 and the last MPEG VVC/H.266. The proposed architecture operates at 450Mhz with 66.997 Kgate and with 47.36 Kbits of RAM spread over two RAMs, one of size 46.08 Kbits for the 7×7 and 5×5 DSF and the second of size 1.28 Kbits for the CCALF filter. The design is able to decode in real-time 4k video at 30fps for 4:2:0 and 4:2:2 Chroma sub sampling formats.

Table 9.11 Comparison of different ALF hardware designs on FPGA and ASIC platforms.

Solutions	Du *et al.* [76]	Conceicao *et al.* [77]	Redies *et al.* [78]	Redies *et al.* [79]	Proposed design
Platform	FPGA 65 nm	FPGA 90nm	FPGA 90nm	FPGA 40 nm	ASIC 28 nm
9×9 DSF	✓	✓	✓	✗	✗
7×7 DSF	✓	✓	✓	✗	✓
5×5 DSF	✓	✓	✓	✓	✓
Gradient sum	✗	✗	✗	✗	✓
CCALF	✗	✗	✗	✗	✓
Chroma sub-sampling	4 : 2 : 0	4 : 2 : 0	4 : 2 : 0	4 : 2 : 0	4 : 2 : 2/4 : 2 : 0
LUTs	15561	1660	2071	113	✗
Gates	✗	✗	✗	✗	66997
Frequency (MHz)	211	115	229	41	450
DSPs/RMs	41	39	20	13	26★
Throughput (fps)	1920 × 1080 30 fps	3840 × 2160 33 fps	2560×1600 49 fps	1920 ×1080 19 fps	3840×2160 30 fps
Memory	✗	✗	✗	✗	47.36 Kb

★ The number of multipliers is provided for information since it is already included in the total number of gates.

9.7 Conclusion

In this chapter a hardware implementation of the ALF has been investigated targeting a real time 4Kp60 VVC decoder on ASIC platforms. The ALF is part of the in-loop carried-out at the end of the decoding loop. It aims at reducing visible artifacts from the reconstructed image by performing adaptive filtering to minimize the mean squared error (MSE) between original and reconstructed samples based on Wiener filtering. The proposed hardware implementation relies on regular multipliers and sustains a constant system latency with a fixed throughput of 2 pixels per cycle. It uses a total of 26 multipliers in a pipelined architecture for all ALF filters including the 7×7, 5×5 and CCALF diamond shape filters. This is the first hardware design that addresses the memory architecture and the gradient computation of the ALF. By creating an alternation between Luma and Chroma components, the memory size is reduced at most, from CTB wise to just minor lines of CTB. The proposed design is able to reach real time decoding of 4K videos with 4:2:2 sub-sampling at 60 frames par second. This design has been successfully integrated in a hardware ASIC decoder supporting recent MPEG standards including AVC, HEVC and VVC. To the best of our knowledge, this is the first complete hardware design supporting all ALF proprieties.

Chapter 10

ALF and ALF decision HW design for 4K VVC encoder

10.1 Introduction

In the Versatile Video Coding (VVC) standard, Adaptive Loop Filter (ALF) with block-based filter adaption is applied for the Luma and Chroma components. For Luma, one among 25 filters is selected for each 4×4 block, based on the direction and activity of local gradients. For Chroma, the same filter is used for all samples of the Coding Tree Block (CTB). As presented in Chapter 8, the ALF uses three Diamond Shape Filters (DSFs) shown in Figure 8.5 and Figure 8.6. The 7×7 DSF, Figure 8.5:(a), is applied on the Luma samples. The 5×5 DSF, Figure 8.5:(b), and 3×4 Cross-Component Adaptive Loop Filter (CCALF), Figure 8.6, are applied for chroma samples. For Luma, there are two types of filter coefficients: 1) fixed filter coefficients, 2) customized filter coefficients signaled in Adaptation Parameter Set (APS). The Fixed filter coefficients for Luma are sixteen sets of filters that are defined by the VVC standard where each set embeds 25 different filters. For the Chroma component, one type of filter coefficients is used which is customized filter coefficients signaled in APS for the both 5×5 and CCALF DSF.

This work focuses on the hardware design of the ALF in the encoder context. In fact, the aim is to design a real time VVC encoder that supports 4Kp60 videos over four cores where each core is able to run 1080p60 videos at a frequency of 450Mhz. To do so, it defines a fixed budget for every 64×64 Coding Tree Unit (CTU). This budget is uniform for the four cores and it equals to *9000* clock cycles defined. This is the main constraint that defines the processing power of the loop filters including the ALF and its decision. Moreover, the architecture of the proposed decision which has been studied in this PhD is as follows. It has been designed after an in-depth study of the ALF decision algorithm and ended with a first generation hardware architecture, yet implementable, able to catch a significant amount of this tool performance. In fact, the study revealed that a good trade-off between rate, distortion and hardware complexity is found when the encoder activates the ALF for Luma samples and for CTUs of size 64. In addition, it limits the decision process on four fixed sets of coefficients, thus, testing four out of sixteen filter set indices defined in the standard. These indices are adaptively selected by a control software depending on the picture type and the Quantization

Parameter (QP) values.

The ALF encoder is applied over two phases: the filtering and decision process. Both phases are only applied for Luma components and for CTUs of size 64. Figure 10.1 shows the number of pixels to process at each phase. As it can be noticed, each phase has 4096 pixels to process. Thus, making a total of 8192 pixels to process for both phases. Taking the cycle budget into account, the ALF encoder should computes at least 1 pixels/cycle at each phase. However, in practice, the performance should have a computation margin to compensate for wait states introduced by control and communication operations. Therefore, one of the phases should be a bit faster to leave a margin of security cycles. Since the decision phase is expected to have more filtering operations in parallel than the second phase, it is more convenient to increase the speed of the filtering phase. Hence, the second phase is constrained with a throughput of 2 pixels/cycle while the decision process remains at a throughput of 1 pixel/cycle.

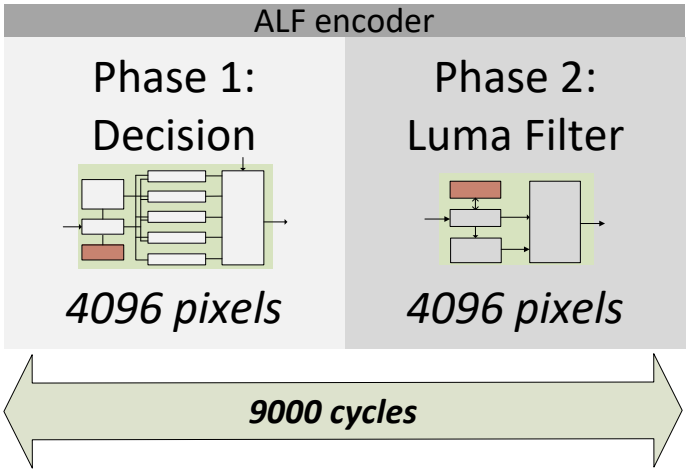


Fig. 10.1 ALF encoder phases

The following sections are organized as follows. Section 10.2 describes the Luma ALF filter for two pixels/cycle throughput. It presents the hardware designs of the memory, gradient sum and filtering modules. Section 10.3 describes the ALF decision architecture for a throughput of one pixel/cycle. It describes the hardware designs of the memory, gradient, filtering and the Sum of Square Differences (SSD) modules. Section 10.4 presents the experimental setup and the synthesis results with a comparison with the ALF decoder and finally, Section 10.5 concludes the chapter.

10.2 ALF for Luma only design

The ALF encoder is activated only for Luma samples and for CTB of size 64. For Chroma samples and for other CTB sizes the module is bypassed. At the input, for each CTB line, the Luma and Chroma blocks are scanned alternatively: one 4×4 Y followed by one 4×4 Cb followed by one 4×4 Y followed by one 4×4 Cr. For 4:2:0, this alternation is applied half of the time. At the output, the ALF module delivers one 4×4 block every 8 cycles whether it is Luma or Chroma. In case Chroma

blocks are provided to the module, they are bypassed with a delay line that has the same latency as if a Luma block is beaning processed. The ALF hardware module should be able to process 2 pixels every one clock cycle which is the main constraint for this module. Table 10.1 summarizes the ALF processor control interface. A positive pulse in alf_enable launches the ALF process. Using the block position defined by *alf_blk_pos_x* and *alf_blk_pos_y*, this module filters the 4×4 input block using the filter index signaled in the interface *alf_filter_index*. With respect to throughput constraint, the input block is refreshed with new data through out the *alf_blk_rec* interface every eight clock cycles. Finally, the ALF module starts generating the results after a fixed system latency and then it continues generating filtered blocks every eight clock cycles without any stall.

Table 10.1 ALF for Luma module interfaces

Signals	I/O	#BITS	Description
clk	I	1	System Clock
rst_n	I	1	System reset, active low
alf_enable	I	1	Activation pulse to start
alf_color_comp	I	2	00:Y, 01:Cr and 10:Cb
alf_filter_index	I	4	0 to 15 indices for fixed coefficients
pic_height	I	11	Picture height
pic_width	I	11	Picture width
alf_blk_rec	I	16×10	Data input interface, 16 samples for 10-bits samples
alf_blk_pos_x	I	5	4x4 block x position in the CTU
alf_blk_pos_y	I	5	4x4 block y position in the CTU
alf_blk_out	O	16×10	Output bus
alf_out_enable	O	1	End of 4x4 block signal

10.2.1 Top level design

Figure 10.2 depicts the top level design of the ALF module. It is composed of four sub-modules:

- *Random-Access Memory (RAM) memory:* it stores the required blocks in order to apply the gradient and the 7×7 Luma filter.
- *Memory handler:* it takes advantage of successive Luma blocks to reduce the total number of memory accesses.
- *Gradient module:* computes the gradient of all direction and determines the classification value. It also embeds two sub-modules: the transpose and the Read-Only Memory (ROM) for fixed coefficients.
- *Filter module:* it applies the 7×7 Luma DSF by using 24 regular multipliers to sustain 2 pixels/cycle throughput. It delivers the final filtered block.

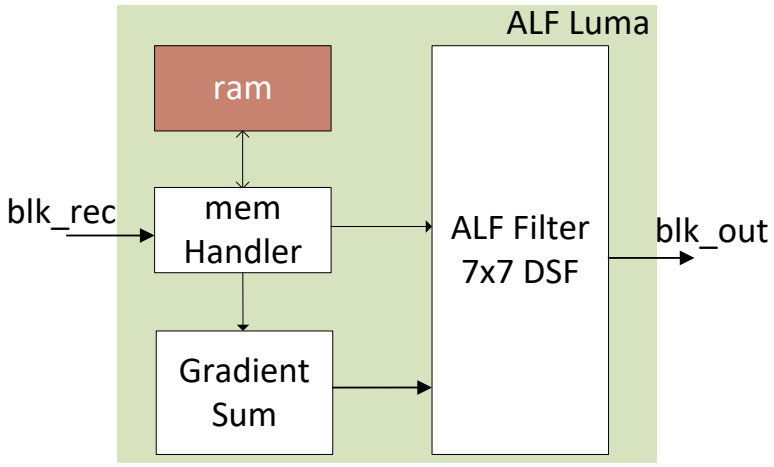


Fig. 10.2 ALF encoder top level design

10.2.2 Memory dimensions and handler

Luma blocks are scanned in raster scan order within the CTB. To compute the gradient and apply the 7×7 DSF, each 4×4 Luma block needs its eight surrounding neighbors (top left (TL), top (T), top right (TR), left (L), right (R), bottom left (BL), bottom (B), bottom right (BR)). Therefore, a distance of one CTB line between the received block and the current block must be conserved in order to filter the 4×4 block, as shown in Figure 10.3. The block in orange is the one to be filtered (Curr) which is filtered only at the reception of the block in purple. This implies that two CTB lines needs to be stored in the memory. Using a rotating memory, these two lines are sufficient to filter the entire Luma CTB. Same as for the ALF decoder, blocks that belong to the last column of the CTB require neighboring blocks belonging to the first column of the neighbor CTB. Since CTBs within the same frame are raster scanned, the last 4×4 column of the CTB (CTBsize-4 : CTBsize) along with its neighboring left 4×4 column (CTBsize-8 : CTBsize-5) is also stored in memory.

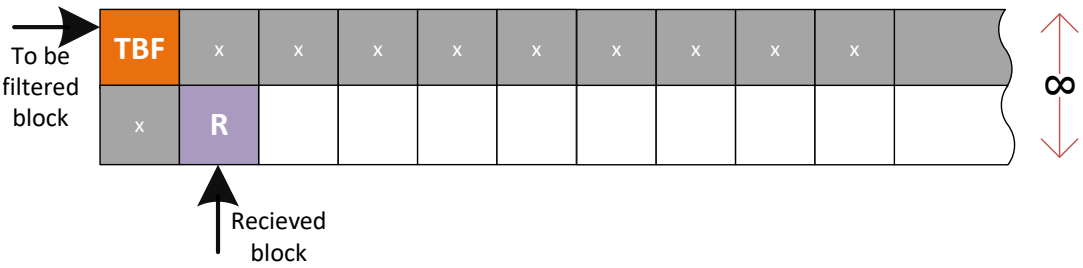


Fig. 10.3 Distance between the received and the to be filtered Luma block

Lines and columns blocks are stored in the same memory and by using offsets columns and lines are defined. The memory dimension is measured on CTBs of size 64, making its total size to 12288-bits of RAM (66 4×4 blocks, for 12-bits) and it is computed as follows:

- CTB lines: two 4×4 lines measured on the CTB of size 64. It is of width 64 pixels which is sixteen 4×4 blocks. For 12-bits pixels, a total of 6144-bits for the CTB lines are stored.
- CTB columns: two 4×4 columns measured on the CTB of size 64. It is of height 64 pixels which is sixteen 4×4 blocks. For 12-bits pixels, a total of 6144-bits for the CTB lines are stored.

On the other hand, the memory handler schedules the writing and reading operations depending on the position of the block within the CTB. For the purpose of retrieving all surrounding blocks and applying all reading and writing operations, the handler needs a total of 9 memory accesses (8 readings+1 writing) for a given input block. These operations require a total of 9 cycles (one operation/cycle) which surpasses the available timing window per block which is 8 cycles (4×4 @2pixels/cycle => 16/2 = 8 cycles). To compensate this, shared blocks between two successive Luma blocks are used, as it was done for the ALF decoder. As it can be seen in Figure 10.4, the surrounding blocks of the block "Curr 0", in green, intersect with the surrounding blocks of the block "Curr 1", in blue, in the shopped region. In order to reduce the memory accesses for the second block, this chopped region is stored in a register memory. Six registers each of size 4×4 (6×(4×4)×12=1152-bits) are used. As a result, the second block will require only three memory operations (2 readings+1 writing). Therefore, for two successive Luma blocks, a window of 16 cycles is allocated where a total of 12 memory operations (10 readings + 2 writings) are needed. These operations are executed in 12 clock cycles. As a result, the memory access timing is aligned with the available window.

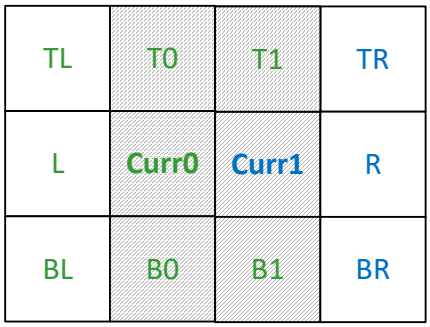


Fig. 10.4 Shared region between two successive Luma blocks

10.2.3 Gradient design

In this module, the gradient of all directions is computed which in turn determines the classification value. Using this value the set of filter coefficients required for the 7×7 DSF are selected. This module is composed of multiple stages as depicted in Figure 10.5.

In the first stage (S1), the module selects the pixels that are used for computing the 1-D subsampled Laplacian for all directions, the selected pixels are stored in the "sel_array" register of size 432-bits (36×12). The results of stage two (S2) are transmitted to the third stage in which the gradient sums are calculated and the classification index (class_value) is deduced. This latter is transmitted to

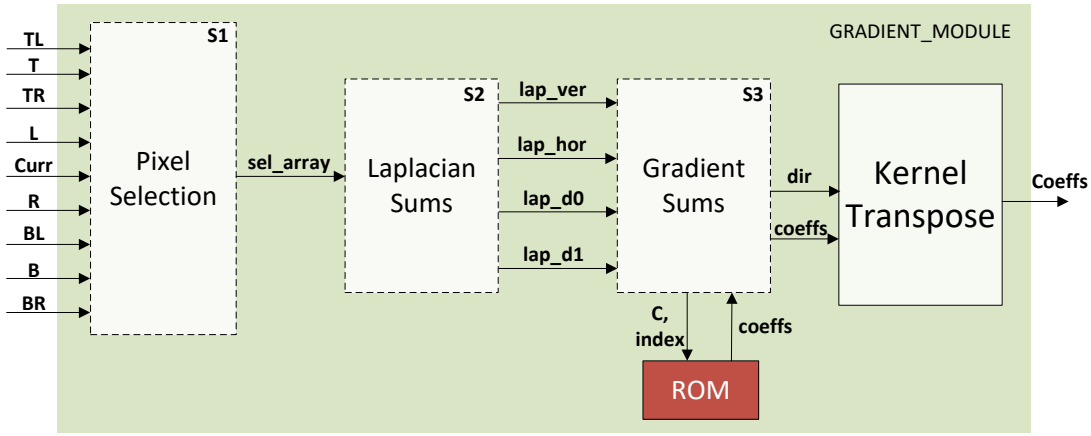


Fig. 10.5 Gradient top level design for Luma filter

the fixed ROM coefficients and to the kernel transpose module in order to retrieve the twelve Luma coefficients. Finally, the filter coefficients are synchronized with the input data, via a delay line, and delivered to the computation module.

10.2.4 7×7 diamond shape filter design

In this module the 7×7 Luma diamond filter is applied. At the input, it receives nine 4×4 blocks containing the "Curr" (to be filtered) block with its eight surrounding neighbors, along with the Luma filter coefficients. At the output, it delivers the filtered block at a throughput of 2 pixels/cycle. Figure 10.6 depicts the top level design of the DSF block. As it can be noticed, this module is decomposed

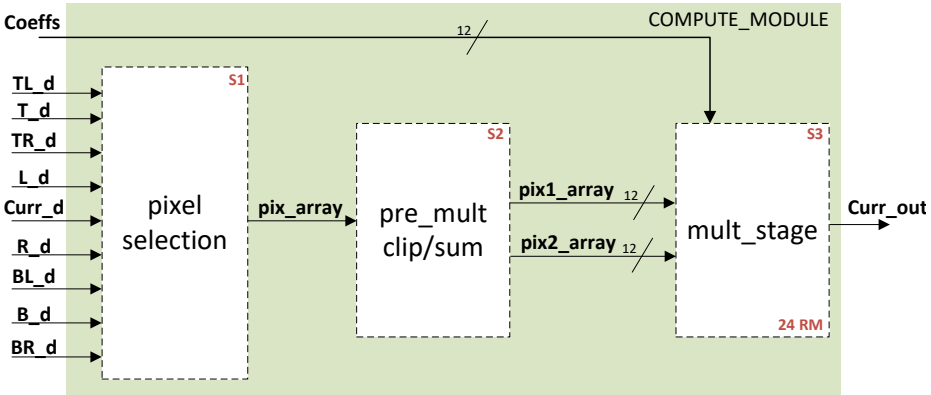


Fig. 10.6 ALF Luma 7×7 filter top design

of three pipelined stages. In the first stage (S1) it selects the pixels that are required for a giving pixel position (x∈[0:3], y∈[0:3]) of the current block. The selected pixels are than transmitted, via the "pix_array" register to the pre multiplication stage (S2), in which clipping and addition operations are applied. The "pix_array" register holds all the pixels that are required in order to filter two

neighboring pixels, as shown in the Figure 10.7. Pixel "12" and "13" are filtered in the same clock cycle, therefore the register array saves all the surrounding samples for both filtering operation. After clipping and addition operations, two separate arrays (pix1_array, pix2_array), each representing one pixel, are delivered to the final stage (S3). In order to process 2 pixels/cycle, this stage uses a total of 24 Regular Multipliers (RMs). Finally, the filtered 4×4 block is outputted.

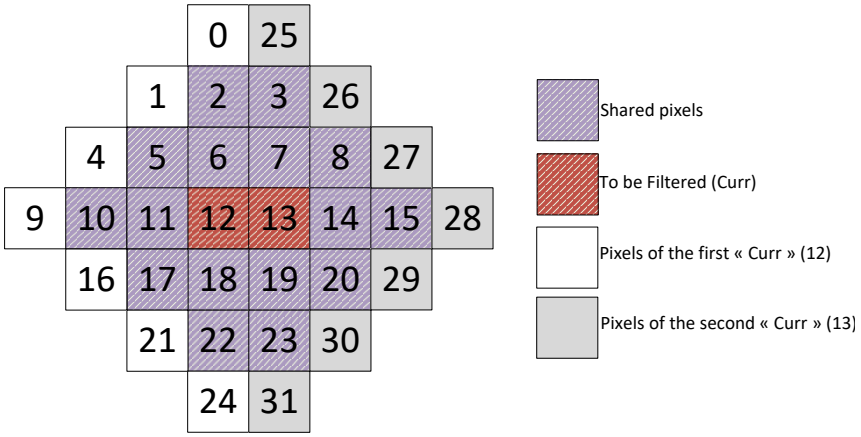


Fig. 10.7 pixel array indexation

10.3 ALF decision design

The ALF decision is performed only for Luma components and for CTBs of size 64. It computes the performance of four filter indices along with bypass. It accesses pixels [0:63]×[0:63] in order to be able to process pixels [0:59]×[0:59]. In order to avoid waiting for CTBs on the right or bottom of the current CTB to make the filter decision, only 60×60 pixels are used in the decision process. Figure 10.8 shows the filtered region of the input CTB. As it can be seen, the decision module receives the CTB, in green, and processes only the red region of size 60×60. In addition, 4×4 blocks are raster scanned within the CTB as shown by the blue arrows. The ALF decision filters each CTB with 4 different filter sets among the 16 fixed ones defined in the standard. It then performs an SSD computation between filtered pixels and source pixels to determine the best filter. The SSD is also computed between the unfiltered and source pixels. For each filter and for filter bypass, a total cost is estimated and the chosen mode is the one that minimizes this cost.

Table 10.2 summarizes the decision processor control interface. A positive pulse in *alf_dec_enable* signal launches the decision process. Using the block position and the four indices values defined by the *alf_dec_filt_idx*, this module executes the filtering operations to the 4×4 input block at a throughput of one pixel per cycle. Therefore, the reconstructed and source input blocks are refreshed with new data every sixteen clock cycles. Unlike the source interface (*alf_blk_src*), the reconstructed data interface (*alf_blk_rec*) holds two 4×4 blocks input, this allows to minimize memory accesses and lower the complexity of the design. This latter is further explained in the coming sections. Finally, the decision module starts generating the results after a fixed system latency and then it continues

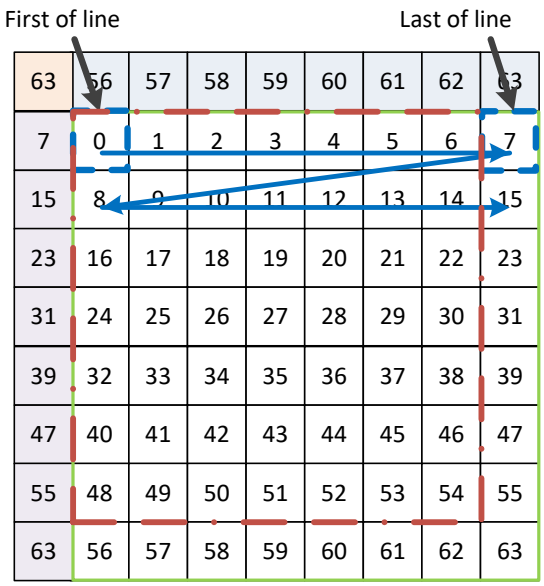


Fig. 10.8 ALF Decision input arrival order and filtered region. Red block is the filtered region while the green block represents the CTB. The blue arrows mimics the sequencing order.

generating decisions every sixteen clock cycles without any stall.

10.3.1 Top level design

Figure 10.9 depicts the ALF decision top-level design. It is composed of five sub-modules:

- *The RAM:* it holds the blocks required to apply the Luma filters.
- *The memory handler:* it communicates with decision memory and retrieves the required blocks for filtering depending on the CTB position within the frame and the 4×4 block position within the CTB.
- *Gradient module:* computes the gradient of all direction and determines the classification value. It also embeds two sub-modules: the transpose and the fixed Luma ROM. Using the filter indices, it determines four different filters that are used by the DSF module.
- *Filter module:* it applies the 7×7 Luma DSF by using 12 regular multipliers to sustain 1 pixels/cycle throughput. This module is duplicated four times. Each one delivers a filtered block with respect to the input filter coefficients.
- *SSD module:* the SSD module computes the sum of square difference between filtered pixels and source pixels to determine the best filter, it also computed the SSD between the unfiltered and source pixels. The module finally delivers the index value that minimizes the SSD cost. In case of filter bypass, it outputs an index value out of range (e.x: “1111”).

Table 10.2 ALF decision design interfaces

Signals	I/O	#BITS	Description
clk	I	1	System Clock
rst_n	I	1	System reset, active low
alf__dec_enable	I	1	Activation pulse to start
alf_color_comp	I	2	00:Y, 01:Cr and 10:Cb
pic_height	I	11	Picture height
pic_width	I	11	Picture width
alf_chroma_subsampling	I	2	"00" : 4:0:0, "01" : 4:2:0, "10" : 4:2:2
alf_dec_filt_idx	I	16	Defines the 4 filter sets that are tested for each CTU: 4 bits -> 1 index
alf_ctu_pos_x	I	11	CTU x position in the frame
alf_ctu_pos_y	I	11	CTU y position in the frame
alf_blk_rec	I	2×16×10	Reconstructed data input interface, two 4 × 4 blocks for 10-bits samples
alf_blk_src	I	16×10	Source data input interface 16 samples for 10-bits samples
alf_blk_pos_x	I	5	4x4 block x position in the CTU
alf_blk_pos_y	I	5	4x4 block y position in the CTU
alf_ctu_filt_idx	O	4	CTU decision index
alf_ctu_filt_idx_ready	O	1	CTU decision ready signal

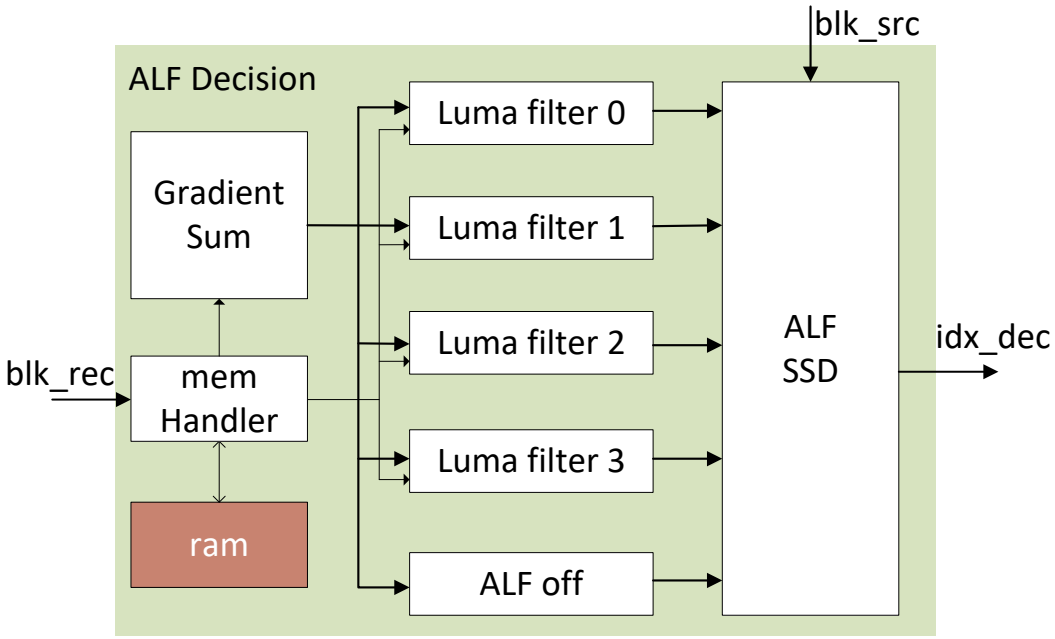


Fig. 10.9 ALF encoder top level design

10.3.2 Memory dimensions and handler

Same as the first ALF module presented in section 10.2, blocks are scanned in raster scan order within the Luma CTB. To compute the gradient and apply the 7×7 DSF, every 4×4 Luma block requires its 8 surrounding as shown in Figure 10.10.

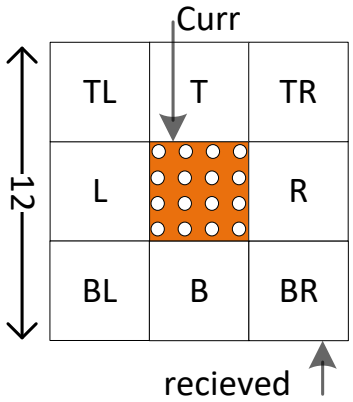


Fig. 10.10 Required blocks for Luma gradient and 7×7 filter

For the ALF decision, and shown in Figure 10.8, only 60×60 samples (red region) of every Luma CTB is processed leaving the 4 rightmost columns and the 4 bottom lines. This eliminates the need to wait for neighboring CTBs in order to finish the current one. As a result, it eliminates the need to store the last column of every CTB. In addition, the input arrival order is critical in order to reduce memory storage. In Figure 10.11, the scanning order for source and reconstructed samples at the input of the decision modules is shown. This latter eliminates completely the need to store source pixels. As it can be noticed, with the reception of the reconstructed blocks, the source blocks of position (x-1, y-1) is scanned. In fact, two 4×4 reconstructed blocks of positions (x-1,y) and (x,y), respectively, are scanned every 16 clock cycles along with one 4×4 source block of position (x-1, y-1). This means that source blocks are utilized directly in the computation of the SSD which eliminates the need to store them. In addition, for every 4×4 block, the available window to retrieve all needed blocks from the memory is aligned with throughput constraint.

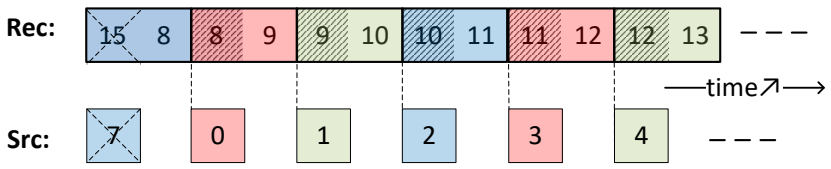


Fig. 10.11 ALF decision source and reconstructed samples sequencing order. Two 4x4 REC blocks are scanned along with one 4x4 SRC blocks every cycle. The shopped blocks belongs to the not filtered region.

Figure 10.12 shows the total number of blocks required for the decision memory. As it can be

noticed, different from the previous modules, blocks at the left and top borders of each CTB require neighboring blocks from left and top previously received CTBs. This is due to the fact that for the decision process, there is no specific clipping operations that are applied at the borders of CTBs. This implies that the last 4 pixel lines of top CTBs are stored in memory. In addition, because CTBs are raster scanned within the frame, a line buffer is used. Since each core of the VVC encoder is expected to support a maximum resolution of 2K, the line buffer should be of size 2048×4-pixles. Along with the line buffer, an additional 4-lines of pixel are stored for CTB of size 64, as shown in orange in Figure 10.12. Using the CTB lines along with a portion of the line buffer in rotating memory, we are able to filter the entire Luma CTB. Finally, the total memory used for the ALF decision for 12-bits samples width is 101.568 Kbits of RAM as shown in Table 10.3.

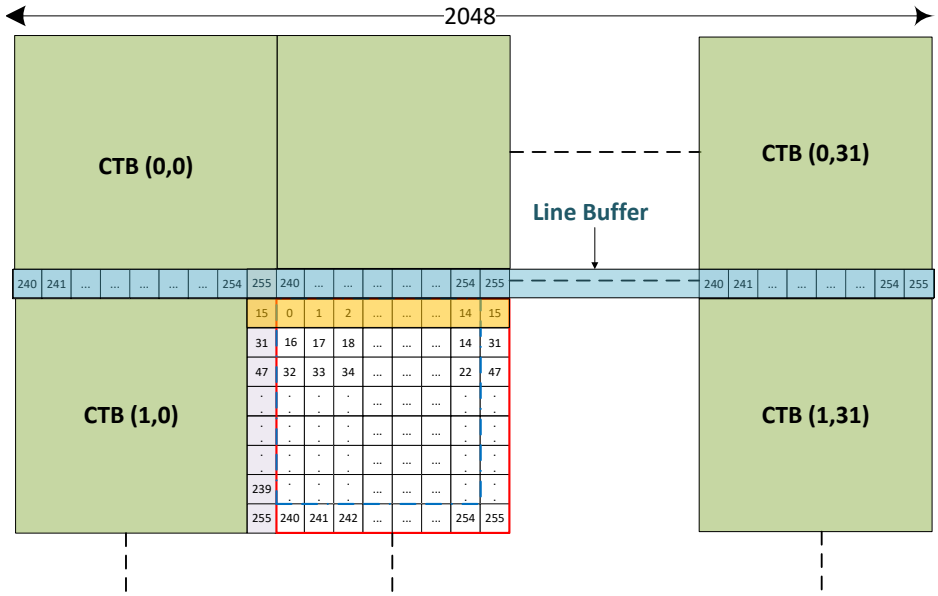


Fig. 10.12 ALF decision RAM blocks

Table 10.3 ALF decision memory cost for different samples width sizes

	Line Buffer (Kbits)	CTB Line (Kbits)	Total (Kbits)
8-bits	65.536	2.176	67.712
10-bits	81.920	2.720	84.640
12-bits	98.304	3.264	101.568

10.3.3 Gradient design

For the ALF decision, four filter indices are signaled to the gradient module. Figure 10.13 depicts the top level of the gradient module. As it can be noticed, same as the previous module, the gradient

computation is performed over four stages. The first stage (S1) selects the required pixels for computing the Laplacian of all directions. The Laplacian is then computed in the second stage (S2). The results of this stage are accumulated in stage 3 (S3) in order to determine the sum of gradients for all directions. Using the result of this latter, the classification value (range: 0 to 24) is determined. Using the class value and the four filter indices, the four Luma filters to be tested are retrieved from the fixed ROM coefficients. In order to respect the throughput of the decision module, the ROM and the geometric transpose modules are duplicated four times as shown in the gradient diagram. Therefore, compared to the gradient of the first ALF module presented in section 10.2.3, this one uses four times more ROM storage making the total ROM cost equals to 24.576 Kbits of ROM.

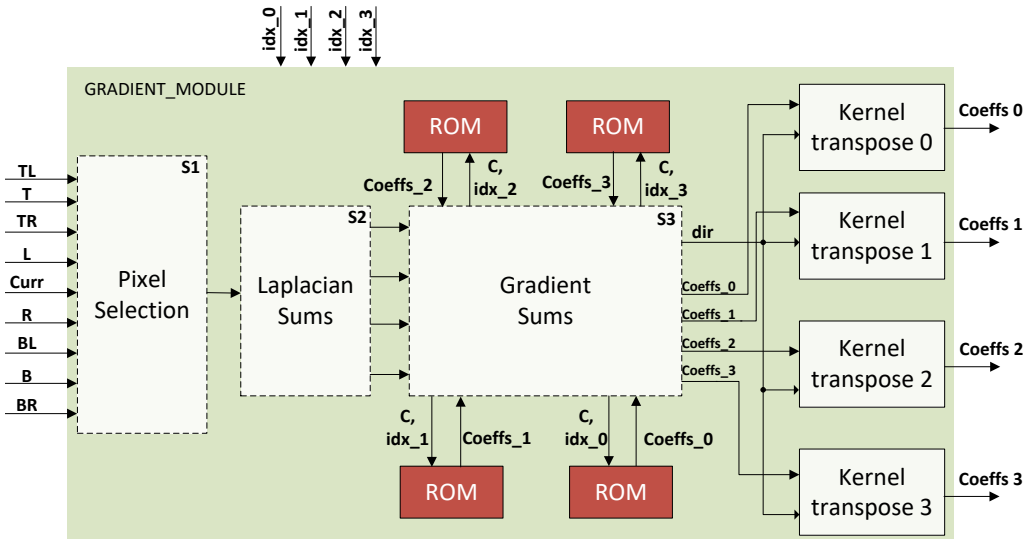


Fig. 10.13 ALF decision gradient sum computation

10.3.4 7×7 diamond shape filter design

In this module the 7×7 Luma diamond filter is considered. At the input, it receives one 4×4 block with its eight surroundings along with the filter coefficients. At the output, it delivers the filtered block. The architecture of this module is similar to the one of the previous module depicted in Figure 10.6. However, since the decision module has a throughput of 1 pixel/cycle, it uses half the number of multipliers in one cycle. Unlike the ALF module, the decision module uses only 12 RM in a pipe-line architecture to filter a 4×4 Luma block. This reduces by half the area cost of the decision design compared to the filtering module. However, the core filter module of the decision is also duplicated four times since it tests four different filters. Therefore, the total cost for this module is still larger compared to the previous ALF core module 10.2.4.

10.3.5 Sum of Square Differences design

In this module, the sum of square differences is computed for each filter and for filter bypass. The SSD is computed for each case between the filtered/unfiltered pixels and the source pixels to determine the best filter. As shown in Figure 10.14, it is performed over three main stages. In the first stage (S1) the SSD of each mode is computed: unfiltered block, filter one, two, three and four. The SSD results are propagated to the second stage (S2) in which the total cost of all indices is computed and regulated with a factor Lamda for every Luma CTB. In the last stage (S3), these modes are put in competition and the one that minimizes the cost is selected and outputted. In case the unfiltered block has the lowest SSD, an index out of range is delivered to the output interface.

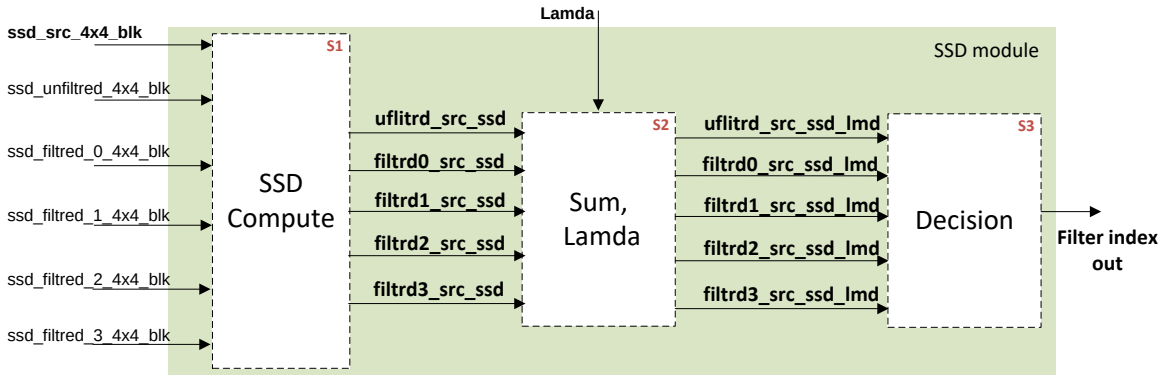


Fig. 10.14 ALF decision SSD module

10.4 Experimental and Synthesis Results

10.4.1 Experimental setup

The proposed design was implemented using VHSIC Hardware Description Language (VHDL). A state-of-the-art logic simulator [66] is used to test the functionality of the ALF memory and core multiplication modules. Unlike the ALF decoder, the setup for the ALF encoder (decision+ALF Luma) is not based on unit tests, instead macro-block testing is applied. This setup is applied with a video encoded stream using the software encoder. At the software encoder side, a portion of code is added that captures the frame status at the input/output of the ALF modules.

- *ALF for Luma test-bench:*

  Figure 10.15 depicts the test-bench top level architecture for ALF module. The Sample Adaptive Offset (SAO) output which is the input frame is directed to the ALF hardware via an input RAM at the target throughput of 2 pixels/cycles. This latter is processed by the ALF hardware module and the results, stored in an output RAM, is compared with the software outputs using self-check techniques.

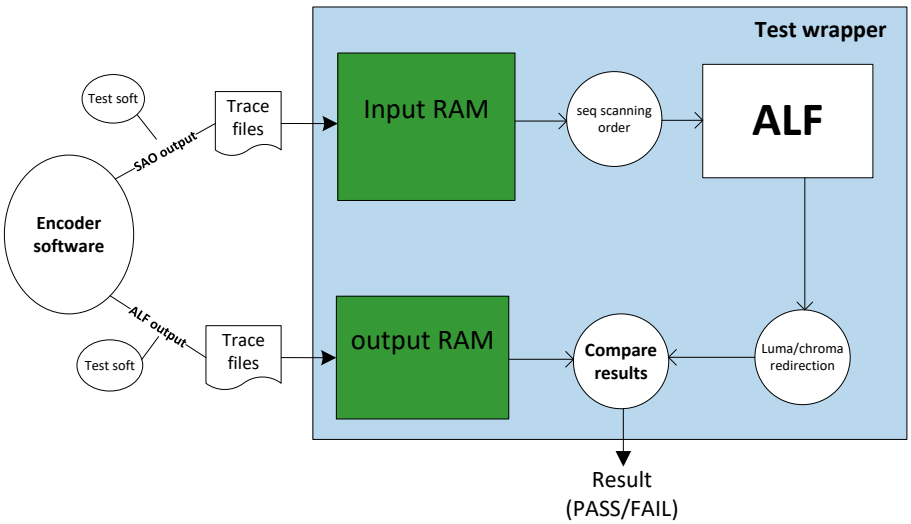


Fig. 10.15 ALF test-bench top level design

- *ALF decision test-bench:*
  Figure 10.15 depicts the test-bench top level architecture for ALF decision modules. Using two input memories, the ALF decision module is feed with source and reconstructed data via the SRC_RAM and REC_RAM, respectively. These data are then processed by the ALF decision module. The final decision index is compared with the decision index computed by the software module. This latter is signaled to the test bench wrapper via a trace file where all the CTU decisions for a given frame are stored.

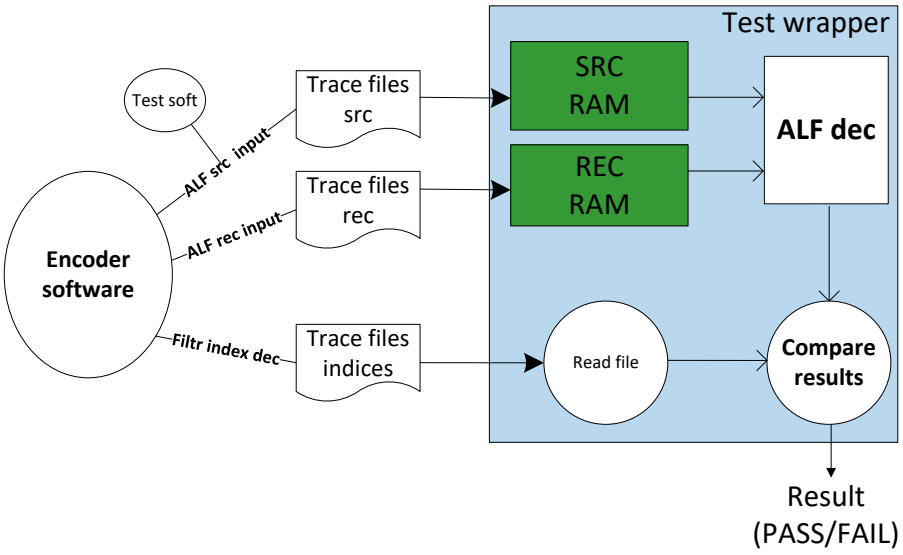


Fig. 10.16 ALF decision test-bench top level design

For both design, a list of video sequences were executed to fully verify the functionality of the

designs. These test sequences are separated by categories and they are shown in Table 10.4. In the

Table 10.4 Test sequences category and status

Test category	Resolutions	nbr. of tests	sample width (bits)	status pass/fail
Small resolution	64×64, 128×128, 256×256	15	8,10, 12	pass
Corner resolution	264×264 to 312×312 by step of 8	60	8,10, 12	pass
Standard resolution	720×(480,567), 1280×720 1920×544, 1920×1080	20	8,10, 12	pass

next sections the synthesis results of the ALF and its decision process are evaluated.

10.4.2 ALF for Luma results

Table 10.5 gives the results of ALF module in terms of chip surface and memory consumption targeting Application-Specific Integrated Circuit (ASIC) platforms at 450 Mhz. The table shows how resources are distributed over the sub-modules of the ALF. The ALF design operates at 450 MHz with a total of 41984 um^2 surface including a total of 12288-bits of RAM used to store two lines of 4×4 macro-blocks for Luma component measured for CTUs of size 64×64. The proposed design uses 24 regular multipliers in a pipelined architecture. The DSF module consumes 29.4% of the total area while embedding the 24 multipliers. The memory access module consumes 37.88% which has the largest size, this is because the module exploits successive neighboring blocks in order to reduce the memory accesses. The gradient module has 28.9% of the total area and it embeds the Fixed ROM and transpose coefficients modules. To further evaluate the performance of the ALF design, two input

Table 10.5 Area synthesis results for ALF for Luma only @2 p/c at 450 MHz on ASIC 28-nm platform for typical condition of 85 Celsius with 0.8 input voltage.

	Total Area (um^2)	Percent Total (%)	Combinational Area (um^2)	Non-combinational Area (um^2)	Memory (Bits)	RM
memory access module	15906	37.88	6171	9735	12288	
7×7 DSF	12360	29.4	7527	4833	0	24
Gradient	12136	28.9	6925	4339	0	
Fixed ROM	529	1.3	364	164	6144	
Transpose coeffs	341	0.8	110	231	0	
ALF top level	41984	100	21197	19692	18432	24

voltages at a typical condition of a temperature of 85 Celsius are used. The power and area results can be seen in Table 10.6. As it can be noticed, for 0.8V the power consumption is about 13.607 mW which is almost 30% larger than the one with the 0.65V which is about 9.214 mW. Reducing the power consumption comes at the expense of larger chip area, when operating at 0.65 V the total area is reduced by 11%.

Table 10.6 Power results for ALF @2 p/c at 450 MHz on ASIC 28-nm platform for typical condition of 85 Celsius

	0.8 V	0.65 V
Power (mW)	13.607	9.214
Total area (um^2)	41984	47028

10.4.3 ALF decision results and analysis

Table 10.7 gives the results of ALF decision module in terms of chip surface and memory consumption targeting ASIC platforms at 450 Mhz. The table shows how resources are distributed over the sub-modules of the ALF decision. It operates at 450 MHz with a total of 58628 um^2 surface including a total of 101568-bits of RAM used to store two lines of 4×4 macro-blocks for Luma component measured for CTUs of size 64×64 and a line buffer for 2K frame size. Unlike the first ALF module, the decision module operates at 1 pixel/cycle throughput which results in the use of 12 multipliers for the its DSF module. For four DSF modules has the largest part of chip area for the decision module which is 52%. It uses a total of 48 RMs in parallel with 12 RMs for each DSF. The memory access module has about 20% of the total area, since it does not require any memory access reduction it has a lower area compared to the DSF and gradient module. The gradient module has 22.5% of the total are while it embeds four duplication of the Fixed ROM and four duplication of the transpose coefficients modules. Finally, the SSD module which computes the sum of square differences has the lowest area percentage which is 5.4%.

Same as the ALF for Luma design, two input voltages at a typical condition of a temperature of 85 Celsius are used to further evaluate the performance of the decision module. The power and area results can be seen in Table 10.8. As it can be seen, for 0.8V, the power consumption is about 17.9 mW which is 34% larger than the one with 0.65V which is about 11.913 mW. This reduction in term of power consumption comes at the expense of larger chip area. In fact, when operating at 0.8 V the total area is reduced by 12%.

10.4.4 ALF decoder VS ALF encoder

Table 10.9 shows a comparison between the performance of the ALF decoder and ALF encoder designs. It evaluates the designs in terms of speed, supported filters, chip area, memory cost and power consumption targeting ASIC platforms at 450 Mhz. As it can be noticed, the ALF decoder implementation supports all CTU sizes and all filters including 7×7, 5×5 DSF and CCALF. On the other hand, the ALF encoder is only applicable for CTUs of sizes 64 and for Luma component only. As previously shown, the ALF encoder has two phases -the filtering and the decision- while on the decoder side there is only the filtering phase. This fact increases the overall consumption of the ALF encoder and makes the total area cost larger by 33% comparing to the decoder cost. Taking into account both the filtering and the decision process for the encoder, the total number of RMs is three times larger than the one for decoder. This fact remains convenient since the decision process duplicates the filtering module in order to test multiple indices. Finally, it can be concluded that the

Table 10.7 Area synthesis results for ALF for Luma only @2 p/c at 450 MHz on ASIC 28-nm platform for typical condition of 85 Celsius with 0.8 input voltage

	Total area (um^2)	Percent total (%)	Combinational trea (um^2)	Non-combinational area (um^2)	Memory (Bits)	RM
Memory access module	11525	19.65	6171	3735	84640	
7×7 DSF 1	7646.5	13	4633	3013	0	12
7×7 DSF 2	7646.5	13	4633	3013	0	12
7×7 DSF 3	7646.5	13	4633	3013	0	12
7×7 DSF 4	7646.5	13	4633	3013	0	12
Gradient	9886	16.9	5439	2204	0	
Fixed ROM 1	530	0.9	365	164	6144	
Fixed ROM 2	530	0.9	365	164	6144	
Fixed ROM 3	530	0.9	365	164	6144	
Fixed ROM 4	530	0.9	365	164	6144	
Transpose coeffs 1	341	0.6	128	231	0	
Transpose coeffs 2	341	0.6	128	231	0	
Transpose coeffs 3	341	0.6	128	231	0	
Transpose coeffs 4	341	0.6	128	231	0	
SSD module	3147	5.45	2447	700	0	
Decision top level	59628	100	34590	21300	109216	48

Table 10.8 Power results for ALF decision @1 p/c at 450 MHz on ASIC 28-nm platform for typical condition of 85 Celsius

	0.8 V	0.65 V
Power (mW)	17.89	11.913
Total area (um^2)	58628	66075

choice of limiting the ALF encoder to Luma components only is very efficient in terms of resources and development complexity. Compared to a full version where it supports all color components, it certainly reduces heavily the total area, memory and power consumption.

10.5 Conclusion

In this chapter a hardware implementation of the ALF encoder supporting the decision and the filtering phase has been investigated targeting a real time 4Kp60 VVC encoder on ASIC platforms. The decision and the filter are applied only for Luma samples and for CTUs of size 64. The decision process computes four filters among sixteen defined by the standard in order to reduce the cost of the encoder designs. Both architectures rely on regular multipliers and sustains a constant system latency with a fixed throughput of 1 pixels per cycle for the decision process, and 2 pixels per cycles for the filtering phase. It uses a total of 24 multipliers in a pipelined architecture for 7×7 DSF Luma and 48

Table 10.9 Comparison between ALF decoder and encoder designs targeting ASIC platform for typical condition of 85 Celsius with 0.8 input voltage

	ALF Decoder	ALF encoder	
		Luma filter	Decision
CTU sizes	16, 32, 64 and 128	64	
Filters	7×7, 5×5 DSF and CCALF	7×7 DSF	
Throughput (pixels/cycle)	2	2	1
Total RMs	26	24	48
RAM (Kbits)	50.560	12.288	84.640
Total area (um^2)	66997	41984	58628
Power (mW)	9.5819	13.607	17.89

multipliers for the decision. This contribution is the first hardware design that addresses ALF for the encoder context. By limiting the ALF for Luma components only and four filter indices, the memory and the area sizes are efficiently reduced. The proposed design is able to reach real time decoding of 4K videos with 4:2:2 sub-sampling at 60 frames par second. This design has been successfully integrated in a hardware ASIC encoder supporting recent MPEG standards including Advanced Video Coding (AVC), High Efficiency Video Coding (HEVC) and VVC. To the best of our knowledge, this is the first complete hardware design study addressing the ALF encoder.

Chapter 11

Conclusions and Perspectives

11.1 Summary of PhD Achievements

Video compression has been a crucial necessity in our life because of the worldwide video contents invasion and the increasing of its digital size. In video streaming, coding is crucial because the compressing of the raw video reduces the bandwidth making it easier to transmit, while maintaining a good quality of experience for end viewers. If all the video content was not compressed, available bandwidth on the Internet would be inadequate to transmit all of it and prevent us from deploying widespread, distributed video playback services. The ability to stream video on multiple devices in our homes, on-the-go using mobile, or even while video chatting with loved ones across the globe, even with low bandwidth, is thanks to video coding. Improving the coding efficiency was always one of the crucial issues of various compression standards that aim to get the most compact representation of the reconstructed video, with a high subjective quality.

Lately, every video coding standard has led to 50% bit-rate reduction compared to its predecessor as a priority on top of its objectives, especially for modern video coding standards such as Advanced Video Coding (AVC), High Efficiency Video Coding (HEVC) and recently Versatile Video Coding (VVC). Nevertheless, parallel to the bit-rate reduction, it is unavoidable that a better coding efficiency comes with the expense of higher complexity and requires more processing power. Moreover, for real time videos and due to the limited amount of parallelism, software-based coding is unable to achieve the maximum quality offered by the encoder standard technology without any delay. This is where the application of real time video encoding is more suitable for dedicated Hardware (HW) designs. Using HW platforms such as Application-Specific Integrated Circuit (ASIC) enables the access for more parallelism capability and higher computational power. A study conducted by Xilinx [11] showed that the use of dedicated HW design through ASIC for real time applications is more efficient in term of achievable quality and power consumption. In this framework, this thesis presented a in-depth studies for multiple tools of the VVC standards. In particular, the transform part, composed of the Multiple Transform Selection (MTS) and Low Frequency Non-Separable Transform (LFNST), and the in-loop filter part, specifically, the Adaptive Loop Filter (ALF). Thought out this study we showed how these tools are crafted and optimized when targeting a real-time ASIC chip for both decoder and encoder contexts. The main contributions of this thesis are summarized in Figure 11.1 and they are

as follows:

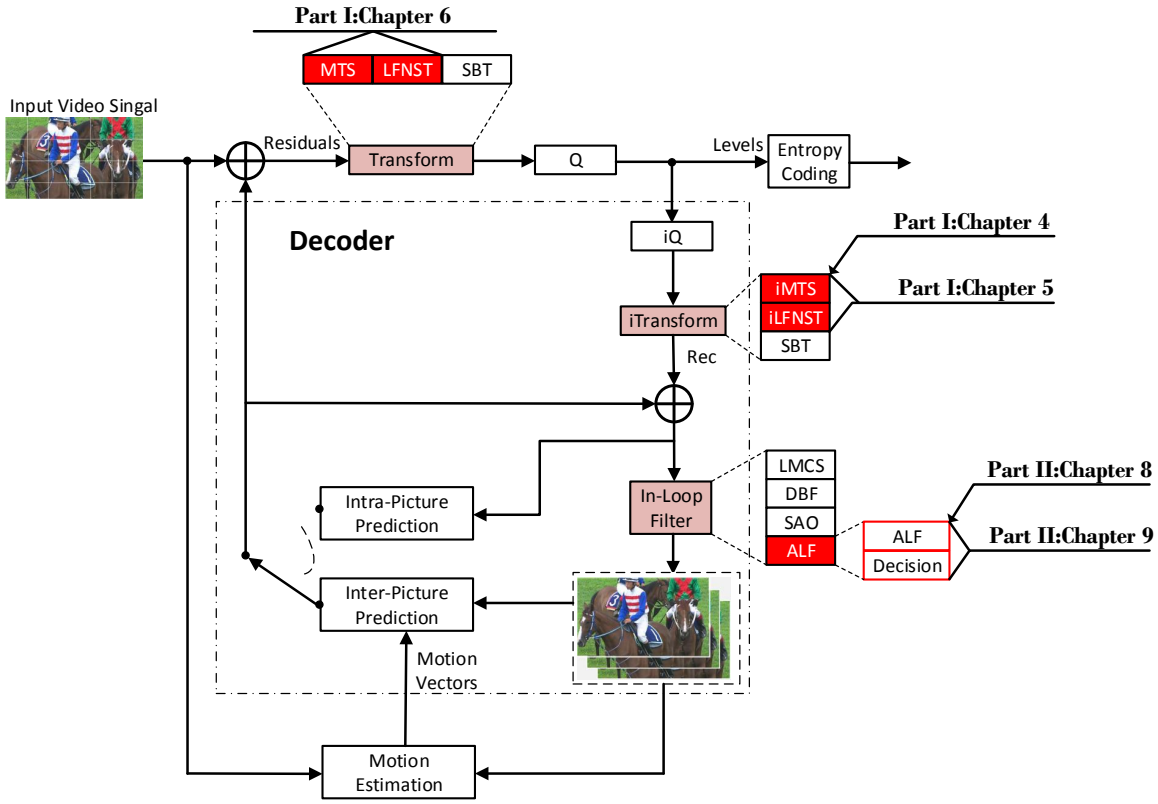


Fig. 11.1 Thesis summary

- Part I introduces the video compression background and its standardization history. It presents a brief overview on the state-of-the-art video coding standard, VVC, explaining his coding scheme and the main blocks of the coding chain. In addition, presents the characterization of a real time video codec with a comparison between several commercial encoders.

- Part II presented the first contribution for an efficient HW implementation of the VVC transform in literature. Chapter 4, presented the background of the MTS and LFNST tools with a comparison between the existing HW implementations of the transforms module. In Chapter 5, a HW implementation of the inverse 1D MTS targeting 4Kp30 resolution and 450Mhz frequency is presented. the proposed design uses 32 shared multi-pliers in a pipe-lined architecture. It consists in a multi-standard architecture that supports the transform block of recent MPEG standards including AVC, HEVC and VVC. The architecture is optimized and removes unnecessary complexities found in other proposed architectures by using regular multipliers instead of constant multipliers. Then, in Chpater 6, for the same decoder context, the LFNST is added to the transform module. The proposed design uses a total of 64 regular multipliers in a pipe-lined architecture. It leverages all primary and secondary transforms optimisations including butterfly decomposition, coefficients zeroing and the inherent linear relationship between

the transform types. The unified transform sustains a constant throughput of 1 sample per cycle and a fixed system latency for all block sizes. It is the first contribution in the literature that includes both the MTS and LFNST in unified architecture for a real-time 4K professional VVC decoder. In addition, Chapter 7 presented a study for a shared design between the Discrete Cosine Transform (DCT)-2 and LFNST and between the DCT-2 and its inverse for the encoder context. The study evaluates and compares the costs of these two shared designs in term of multiplications, additions and area cost.

- Part III presented the first contribution for an efficient HW implementation of the ALF tool in literature. Chapter 8 presents the background of the ALF. It shows the different filters used by the ALF for the Luma and Chroma components. In Chapter 9, a novel HW-architecture for the ALF filter, considering its three filters: 7×7, 5×5 and 3×4 Cross-Component Adaptive Loop Filter (CCALF) is presented. The proposed solution establishes a scanning order between Luma and Chroma components that reduces significantly the memory. The design takes advantage of all ALF features and establishes an unified HW module for all the filters. It uses 26 regular multipliers in a pipe-lined architecture with a fixed throughput of 2 pixels/cycle and a fixed system latency regardless of the selected filter. In Chapter 10, this module has been studied and implemented for the encoder context. In this context, the ALF has two phases: the filtering and the decision process. It has been designed after an in-depth study of the ALF decision algorithm and ended with a first generation HW architecture, yet implementable, able to catch a significant amount of this tool performance. In fact, the study revealed that a good compromise between rate, distortion and HW complexity is found when the encoder activates the ALF for Luma samples and for Coding Tree Unit (CTU)s of size 64. In addition, It limits the decision process on four fixed sets of coefficients, thus, testing four out of sixteen filter set indices defined in the standard. To the best of our knowledge, this is the first contribution in the literature that proposes a full HW design study for the ALF module in the context of encoder and decoder.

Finally, the proposed designs have been integrated into a real-time professional HW video encoder/decoder.

11.2 Perspectives for future works

In part II, we presented a novel HW architecture for the VVC transform part. The proposed design supports the MTS and LFNST tools. It leverages all primary and secondary transforms optimisations including butterfly decomposition, coefficients zeroing and the inherent linear relationship between the transform types. Although it unifies both tools in the same design, the core multiplications module remains separable for each design where each tool uses 64 Regular Multipliers (RMs), thus, 128 RMs for the unified transform. As a future work, it is possible that the core multiplication module could be unified for both the MTS and LFNST, by using 64 RMs for a unified core multiplication. This may cost an increase in the Read-Only Memory (ROM), however, it could reduce the overall area of the transform design. It is worth noting that this shared design could be exploited for the encoder context by embedding the LFNST with the other trigonometric transform types. In addition, our proposed design relies entirely on RM-based architecture, however, many state of the art works proposes a very

low cost designs when using Constant Multipliers (CM), especially for the DCT-2 transform type. As a future work, the DCT-2 of the MTS could implemented separately using an optimal CM-based architecture while the remaining MTS types (DCT-8 and DST-7) and the LFNST are implemented using RM-based architecture. Consequently, a well studied architecture could put an end to the rivalry between the RM and CM-based architectures when put in competition with our proposed design. On another note, the encoder transformation could take advantage of machine learning algorithms to eliminate the exhaustive search for the best partitioning and type of transformation. The challenge here lies in creating a HW design for the selected machine learning algorithm. Then this letter could be compared to the classic solution and it could steer the future of the HW encoder in a new direction.

In part III, we presented the first HW architecture in literature for VVC ALF targeting 4Kp60 resolution for real-time decoder and encoder. For the decoder, we presented a novel scanning order between Luma and Chroma components that allows for a low memory consumption. In addition, the core multiplication part was designed to unified for all ALF filters including 7×7, 5×5 Diamond Shape Filter (DSF) and CCALF. This by taking advantage of the fact that all these filters could be embedded, in term of size, within the 7×7 filter size. For the encoder, the ALF decision and filtering stages were designed based on the study of its algorithm which led to reduce the Chroma filters while catching a significant amount of its performance. As a future work, the ALF encoder could be extended to support one of the Chroma filters. This letter, could be then studied for HW and put in competition with our proposed design. This could reveal the impact of the tool on the coding efficiency and its cost on the HW circuit in term of size, power and memory.

Part IV

References and Publications

References

[1] "HEVC Video Codecs MSE Comparison," in *[Online]:http://www.compression.ru/video/codec comparison/hevc 2018*, 2018.

[2] Z. Zhang, X. Zhao, X. Li, L. Li, Y. Luo, S. Liu, and Z. Li, "Fast dst-vii/dct-viii with dual implementation support for versatile video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 1, pp. 355–371, 2021.

[3] B. Bross, J. Chen, S. Liu, and Y.-K. Wang, "Algorithm description for versatile video coding and test model 8 (vtm 8)," oint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, April 2020.

[4] J. Chen, Y. Ye, and S. H. Kim, "Algorithm description for versatile video coding and test model 8 (vtm 8)," JVET AHG report: Test model software development (AHG3), January 2020.

[5] W.-J. Chien and J. Boyce, "Jvet ahg report: Tool reporting procedure (ahg13)," JVET document T0013, 19th meeting by teleconference, July 2020.

[6] X. L. K. Sühring, "Jvet common test conditions and software reference configurations," JVET Document JVET-H1010, November 2017.

[7] "Cisco Visual Networking Index: Forecast and Trends,," in *[Online]:https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html*, 2018.

[8] "Ericsson Mobility Report," in *[Online]:https://www.ericsson.com/4ad7e9/assets/local/reports-papers/mobility-report/documents/2021/ericsson-mobility-report-november-2021.pdf*, 2021.

[9] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[10] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm, "Overview of the versatile video coding (vvc) standard and its applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[11] O. Gunasekara, "Live video encoding using new aws f1 acceleration the benefits of xilinx fpga for live video encoding," in *F1 instances*, 2017.

[12] F. Bossen, J. Boyce, K. Suehring, X. Li, and V. Seregin, "JVET common test conditions and software reference configurations for SDR video," in *JVET-20th Meeting, by teleconference*, 2020.

[13] A. Segall, E. François, W. Husak, S. Iwamura, and D. Rusanovskyy, "JVET common test conditions and evaluation procedures for HDR/WCG video," in *JVET-20th Meeting, by teleconference*, 2020.

[14] P. Hanhart, J. Boyce, K. Choi, and J.-L. Lin, "JVET common test conditions and evaluation procedures for 360° video," in *JVET-12th Meeting, Macau*, 2018.

[15] Y.-H. Chao, Y.-C. Sun, J. Xu, and X. Xu, "VTM common test conditions and software reference configurations for non-4:2:0 colour formats," in *JVET-20th Meeting, by teleconference*, 2021.

[16] T. Nguyen, T.-C. Ma, and A. Nalci, "JVET common test conditions and software reference configurations for lossless, near lossless, and mixed lossy/lossless coding," in *JVET-17th Meeting, by teleconference*, 2020.

[17] A. Browne, T. Ikai, D. Rusanovskyy, M. G. Sarwer, and X. Xiu, "Common test conditions for high bit depth and high bit rate video coding," in *JVET-21th Meeting, by teleconference*, 2021.

[18] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[19] G. Bjøntegaard, "Calculation of average psnr differences between rd-curves," 2001.

[20] "Video codec for audio visual services at p×64 kbits," *ITU-T. Recommendation*, 1993.

[21] "Generic coding of moving pictures and associated audio information–part2: Video," *ISO/IEC. International Standard 13818-2*, 1999.

[22] "Transmission of non-telephone signals," *ITU-T. Recommendation H.262*, 1999.

[23] "Video coding for low bit rate communication," *ITU-T. Recommendation H.263*, 1995.

[24] "Coding of audio-visual objects–part2: Visual," *ISO/IEC. International Standard 14496-2*, 1995.

[25] "Advanced video coding," *ITU-T. Recommendation H.264*, 2003.

[26] "High efficiency video coding," *ITU-T. Recommendation H.265*, 2013.

[27] A. S. et al, "Joint call for proposals on video compression with capability beyond hevc," *JVET document H1002 (JVET-H1002v4), 8th JVET Meeting*, Oct 2017.

[28] "Versatile video coding," *ITU-T. Recommendation H.266*, 2020.

[29] B. Bross, J. Chen, J.-R. Ohm, G. J. Sullivan, and Y.-K. Wang, "Developments in international video coding standardization after avc, with an overview of versatile video coding (vvc)," *Proceedings of the IEEE*, vol. 109, no. 9, pp. 1463–1493, 2021.

[30] M. Wien and B. Bross, "Versatile video coding – algorithms and specification," in *2020 IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pp. 1–3, 2020.

[31] I. Farhat, W. Hamidouche, A. Grill, D. Ménard, and O. Déforges, "Lightweight hardware transform design for the versatile video coding 4k asic decoders," *IEEE Transactions on Consumer Electronics*, vol. 67, no. 4, pp. 329–340, 2021.

[32] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, "Block partitioning structure in the hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1697–1706, 2012.

[33] B. Bross, J. Chen, S. Liu, and Y.-K. Wang, "Versatile Video Coding Editorial Refinements on Draft 12," in *20th Meeting, by teleconference*, 2020.

[34] M. Saldanha, G. Sanchez, C. Marcon, and L. Agostini, "Complexity analysis of vvc intra coding," in *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 3119–3123, 2020.

[35] X. Zhao, S.-H. Kim, Y. Zhao, H. E. Egilmez, M. Koo, S. Liu, J. Lainema, and M. Karczewicz, "Transform coding in the vvc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3878–3890, 2021.

[36] M. Koo, M. Salehifar, J. Lim, and S.-H. Kim, "Low frequency non-separable transform (lfnst)," in *2019 Picture Coding Symposium (PCS)*, pp. 1–5, 2019.

[37] H. Schwarz, T. Nguyen, D. Marpe, T. Wiegand, M. Karczewicz, M. Coban, and J. Dong, "Improved quantization and transform coefficient coding for the emerging versatile video coding (vvc) standard," in *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 1183–1187, 2019.

[38] H. Schwarz, M. Coban, M. Karczewicz, T.-D. Chuang, F. Bossen, A. Alshin, J. Lainema, C. R. Helmrich, and T. Wiegand, "Quantization and entropy coding in the versatile video coding (vvc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3891–3906, 2021.

[39] M. Karczewicz, N. Hu, J. Taquet, C.-Y. Chen, K. Misra, K. Andersson, P. Yin, T. Lu, E. François, and J. Chen, "Vvc in-loop filters," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3907–3925, 2021.

[40] T. Lu, F. Pu, P. Yin, S. McCarthy, W. Husak, T. Chen, E. Francois, C. Chevance, F. Hiron, J. Chen, R.-L. Liao, Y. Ye, and J. Luo, "Luma mapping with chroma scaling in versatile video coding," in *2020 Data Compression Conference (DCC)*, pp. 193–202, 2020.

[41] A. Norkin, G. Bjontegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, and G. Van der Auwera, "Hevc deblocking filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1746–1754, 2012.

[42] C.-M. Fu, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park, and W.-J. Han, "Sample adaptive offset in the hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, 2012.

[43] L. Li, Y. Song, T. Ikenaga, and S. Goto, "A cabac encoding core with dynamic pipeline for h.264/avc main profile," in *APCCAS 2006 - 2006 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 760–763, 2006.

[44] D. Karwowski, "Precise probability estimation of symbols in vvc cabac entropy encoder," *IEEE Access*, vol. 9, pp. 65361–65368, 2021.

[45] P. Haase, S. Matlage, H. Kirchhoffer, C. Bartnik, H. Schwarz, D. Marpe, and T. Wiegand, "State-based multi-parameter probability estimation for context-based adaptive binary arithmetic coding," in *2020 Data Compression Conference (DCC)*, pp. 163–172, 2020.

[46] X. Zhao, J. Chen, M. Karczewicz, A. Said, and V. Seregin, "Joint Separable and Non-Separable Transforms for Next-Generation Video Coding," *IEEE Transactions on Image Processing*, vol. 27, pp. 2514–2525, May 2018.

[47] X. Zhao, L. Li, Z. Li, X. Li, and S. Liu, "Coupled primary and secondary transform for next generation video coding," in *2018 IEEE Visual Communications and Image Processing (VCIP)*, pp. 1–4, 2018.

[48] F. Bossen, X. Li, A. Norkin, and K. Sühring, "Jvet-o0003," JVET AHG report: Test model software development (AHG3), July. 2019.

[49] M. Budagavi, A. Fuldseth, G. Bjøntegaard, V. Sze, and M. Sadafale, "Core transform design in the high efficiency video coding (hevc) standard," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 1029–1041, 2013.

[50] X. Zhao, J. Chen, A. Said, V. Seregin, H. E. Egilmez, and M. Karczewicz, "Nsst: Non-separable secondary transforms for next generation video coding," in *2016 Picture Coding Symposium (PCS)*, pp. 1–5, 2016.

[51] M. Koo, M. Salehifar, J. Lim, and S.-H. K. and, "Low Frequency Non-Separable Transform (LFNST)," in *Picture Coding Symposium (PCS)*, November 2019.

[52] S. Shen, W. Shen, Y. Fan, and X. Zeng, "A Unified 4/8/16/32-Point Integer IDCT Architecture for Multiple Video Coding Standards," in *2012 IEEE International Conference on Multimedia and Expo*, pp. 788–793, July 2012.

[53] J. Zhu, Z. Liu, and D. Wang, "Fully pipelined dct/idct/hadamard unified transform architecture for hevc codec," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pp. 677–680, May 2013.

[54] P. Meher, S. Park, B. Mohanty, K. Lim, and C. Yeo, "Efficient integer dct architectures for hevc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, pp. 168–178, Jan 2014.

[55] M.Chen, Y. Zhang, and C. Lu, "Efficient architecture of variable size HEVC 2D-DCT for FPGA platforms," *Int. J. of Electronics and Communications*, vol. 73, pp. 1–8, March 2017.

[56] A. Ahmed and M. Shahid, "N Point DCT VLSI Architecture for Emerging HEVC Standard," *VLSI Design*, pp. 1–13, 2012.

[57] A. Mert, E. Kalali, and I. Hamzaoglu, "High Performance 2D Transform Hardware for Future Video Coding," *IEEE Transactions on Consumer Electronics*, vol. 62, May 2017.

[58] M. Garrido, F. Pescador, M. Chavarrias, P. Lobo, and C. Sanz, "A High Performance FPGA-based Architecture for Future Video Coding Adaptive Multiple Core Transform," *IEEE Transactions on Consumer Electronics*, March 2018.

[59] M. J. Garrido, F. Pescador, M. ChavarrÃas, P. J. Lobo, and C. Sanz, "A 2-d multiple transform processor for the versatile video coding standard," *IEEE Transactions on Consumer Electronics*, vol. 65, pp. 274–283, Aug 2019.

[60] A. Kammoun, S. B. Jdidia, F. Belghith, W. Hamidouche, J. F. Nezan, and N. Masmoudi, "An optimized hardware implementation of 4-point adaptive multiple transform design for post-HEVC," in *2018 4th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, pp. 1–6, March 2018.

[61] A. Kammoun, W. Hamidouche, F. Belghith, J. Nezan, and N. Masmoudi, "Hardware Design and Implementation of Adaptive Multiple Transforms for the Versatile Video Coding Standard," *IEEE Transactions on Consumer Electronics*, vol. 64, October 2018.

[62] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, C. Sanz, and P. Paz, "An fpga-based architecture for the versatile video coding multiple transform selection core," *IEEE Access*, vol. 8, pp. 81887–81903, 2020.

[63] Y. Fan, Y. Zeng, H. Sun, J. Katto, and X. Zeng, "A pipelined 2d transform architecture supporting mixed block sizes for the vvc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2019.

[64] Intel-FPGA-Integer-Arithmetic-IP-Cores-User-Guide, "Intel 2017." [Onine]_ Available: https://www.altera.com/en-US/pdfs/literature/ug/ug-lpm-alt-mfug.pdf.

[65] A. Kammoun, W. Hamidouche, P. Philippe, O. Déforges, F. Belghith, N. Masmoudi, and J. Nezan, "Forward-inverse 2d hardware implementation of approximate transform core for the vvc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2019.

[66] "RivieraPro-Aldec-Functional-Verification-Tool," in *[Online]:https://www.aldec.com/en/products/functional_verification/riviera-pro*, 2017-05-29.

[67] I. Farhat, W. Hamidouche, A. Grill, D. Menard, and O. Déforges, "Lightweight hardware implementation of vvc transform block for asic decoder," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1663–1667, 2020.

[68] "Design Compiler Graphical," in *[Online]:https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html*, 2015.

[69] F. Bossen, J. Boyce, K. Suehring, X. Li, and V. Seregin, "JVET common test conditions and software reference configurations for SDR video," in *JVET document K1010 (JVET-K1010)*, July 2018.

[70] G. Bjøntegaard, "Calcuation of Average PSNR Differences Between RD-curves," in *VCEG-M33 ITU-T Q6/16, Austin, TX, USA, 2-4 April*, 2001.

[71] G. Bjøntegaard, "Improvements of the BD-PSNR model," *ITU-T SG16 Q*, vol. 6, p. 35, 2008.

[72] A. Wieckowski, J. Ma, H. Schwarz, D. Marpe, and T. Wiegand, "Fast partitioning decision strategies for the upcoming versatile video coding (vvc) standard," in *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 4130–4134, 2019.

[73] T. Amestoy, A. Mercat, W. Hamidouche, D. Menard, and C. Bergeron, "Tunable vvc frame partitioning based on lightweight machine learning," *IEEE Transactions on Image Processing*, vol. 29, pp. 1313–1328, 2020.

[74] C. Tsai, C. Chen, T. Yamakage, I. S. Chong, Y. Huang, C. Fu, T. Itoh, T. Watanabe, T. Chujoh, M. Karczewicz, and S. Lei, "Adaptive loop filtering for video coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 934–945, 2013.

[75] C. Chen and al, "The adaptive loop filtering techniques in the hevc standard," *Proceedings of SPIE Applications of Digital Image Processing*, vol. 7, no. 35, p. 849913, 2012.

[76] J. Du and L. Yu, "A parallel and area-efficient architecture for deblocking filter and adaptive loop filter," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pp. 945–948, 2011.

[77] R. Conceição, F. Rediess, B. Zatt, M. Porto, and L. Agostini, "Configurable hardware design for the hevc-based adaptive loop filter," in *2014 IEEE 5th Latin American Symposium on Circuits and Systems*, pp. 1–4, 2014.

[78] F. Rediess, L. Agostini, C. Cristani, P. Dall'Oglio, and M. Porto, "High throughput hardware design for the adaptive loop filter of the emerging hevc video coding," in *2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 1–5, 2012.

[79] F. Rediess, C. Cristani, P. Dall'Oglio, M. Porto, and L. Agostini, "Architectural design for the adaptive loop filter of the emerging high efficiency video coding standard," in *not knowing*, pp. 1–5, 2011.

[80] X. Wang, H. Sun, J. Katto, and Y. Fan, "A hardware architecture for adaptive loop filter in vvc decoder," in *2021 IEEE 14th International Conference on ASIC (ASICON)*, pp. 1–4, 2021.

[81] M. Budagavi, V. Sze, and M. Zhou, "Hevc alf decode complexity analysis and reduction," in *2011 18th IEEE International Conference on Image Processing*, pp. 733–736, 2011.

[82] A. M. Kotra, S. Esenlik, B. Wang, H. Gao, and E. A. (Huawei), "Rounding correction for alf virtual boundary processing," JVET AHG report: Test model software development (AHG16), April 2020.

[83] N. Hu, V. Seregin, and M. K. (Qualcomm), "Line buffer problem of cc-alf for 4:2:2 and 4:4:4 sequences," JVET AHG report: Test model software development (AHG16), April 2020.

[84] Y. Wang, L. Zhang, H. Liu, K. Zhang, and Z. D. (Bytedance), "A fix on chroma alf virtual boundary position," JVET AHG report: Test model software development (AHG16), April 2020.

[85] X. M. (PKU), X. Z. (DJI), S. Wang, and S. M. (PKU), "Ccalf virtual boundary issue for 4:4:4 and 4:2:2 format," JVET AHG report: Test model software development (AHG16), April 2020.

[86] J. Chen, Y. Ye, and S. H. Kim, "Algorithm description for versatile video coding and test model 10 (vtm 10)," JVET AHG report: Test model software development (AHG3), October 2020.

Appendix A

Personnel Publications

A.1 Scientific journals

[J1] I. Farhat, W. Hamidouche, A. Grill, D. Ménard and O. Déforges, "Adaptive Loop Filter Hardware Design for 4K ASIC VVC Decoders," in IEEE Transactions on Consumer Electronics, doi: 10.1109/TCE.2022.3146272.
[J2] I. Farhat, W. Hamidouche, A. Grill, D. Ménard and O. Déforges, "Lightweight Hardware Transform Design for the Versatile Video Coding 4K ASIC Decoders," in IEEE Transactions on Consumer Electronics, vol. 67, no. 4, pp. 329-340, Nov. 2021, doi: 10.1109/TCE.2021.3126549.

A.2 International Conference

[C1] I. Farhat, W. Hamidouche, A. Grill, D. Menard and O. Déforges, "Lightweight Hardware Implementation of VVC Transform Block for ASIC Decoder," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020, pp. 1663-1667, doi: 10.1109/ICASSP40776.2020.9054281.

-

Titre: Conceptions matérielles efficaces pour des nouveaux outils du standard VVC ciblant les plates-formes ASIC

Mot clés : Compression vidéo, VVC, implémentation temps réel, optimisation, LFNST, MTS, ALF, ASIC

Resumé : La nouvelle norme de codage vidéo MPEG/ITU appelée Versatile Video Coding (VVC) a été finalisée en juillet 2021. VVC permet une efficacité de codage 40 % supérieure à celle de son prédécesseur, la norme HEVC. Ce gain de codage est apporté par plusieurs outils de codage. Le MTS et le LFNST sont l'un des principaux outils de codage qui ont été introduits dans le VVC pour la partie transformation. En outre, afin d'améliorer la qualité perceptive des trames codées, le VVC ajoute un nouvel outil à son processus de filtrage en boucle appelé ALF. Dans cet élément, des implémentations matérielles pour tous les outils ci-dessus ciblant les plateformes ASIC sont proposées. Elles comportent toutes les optimisations nécessaires pour réduire leur coût en termes de surface de puce, de mémoire et de consommation d'énergie. Toutes les solutions proposées ont été intégrées dans un encodeur et décodeur VVC professionnel en temps réel. La première contribution est une implémentation matérielle efficace de la transformation MTS. Elle consiste en une architecture multistandard qui prend en charge le bloc de transformation des normes MPEG récentes, notamment AVC, HEVC et VVC. Elle est optimisée et supprime les complexités inutiles que l'on trouve dans d'autres architectures proposées en utilisant des multiplicateurs réguliers au lieu de multiplicateurs constants. Ensuite, pour le même contexte, le LFNST est ajouté au module de transformation. Il exploite toutes les optimisations des transformées primaires et secondaires, y compris la décomposition papillon, la remise à zéro des coefficients et la relation linéaire inhérente entre les types de transformées. La deuxième contribution consiste en une implémentation matérielle efficace de l'outil ALF. Pour le décodeur, la solution proposée établit un nouvel ordre de balayage entre les composantes Luma et Chroma qui réduit considérablement la mémoire. En outre, ce module a été étudié pour le contexte du codeur. Dans ce contexte, l'ALF comporte deux phases : le filtrage et le processus de décision. Afin de réduire la consommation de ressources, l'encodeur active l'ALF uniquement pour les échantillons Luma et pour les CTUs de taille 64. Cette dernière solution s'est avérée efficace et a été intégrée dans l'encodeur VVC en temps réel.

Title: Efficient hardware designs of the new Versatile Video Coding (VVC) tools targeting ASIC platforms

Keywords : Video compression, VVC, real-time implementation, optimization, LFNST, MTS, ALF, ASIC

Abstract: The new MPEG/ITU video coding standard named Versatile Video Coding (VVC) was Finalized in July 2021. VVC enable 40% better coding efficiency than its predecessor HEVC standard. This coding gain is brought by several coding tools. The MTS and LFNST are one of the key coding tools that have been introduced in VVC for the transform part. In addition, in order to enhance the perceptually quality of the coded frames, VVC adds a new tool to its in-loop filter process called the ALF. In this element, a hardware implementations for all the above tools targeting ASIC platforms are proposed. They feature all optimisations in order to reduces their cost in term of chip surface, memory and power consumption. All the proposed solutions were integrated in a real-time professional VVC encoder and decoder. The first contribution is an efficient hardware implementation of the MTS transform. It consists in a multi-standard architecture that supports the transform block of recent MPEG standards including AVC, HEVC and VVC. The architecture is optimized and removes unnecessary complexities found in other proposed architectures by using regular multipliers instead of constant multipliers. Then, for the same context, the LFNST is added to the transform module. It leverages all primary and secondary transforms optimisations including butterfly decomposition, coefficients zeroing and the inherent linear relationship between the transform types. The second contribution consist in an efficient hardware implementation of the ALF tool. For the decoder, the proposed solution establishes a novel scanning order between Luma and Chroma components that reduces significantly the memory. In addition, this module has been studied for the encoder context. In this context, the ALF has two phases: the filtering and the decision process. In order to reduce resource consumption, the encoder activates the ALF only for Luma samples and for CTUs of size 64. This latter proved to be efficient and got integrated in the real-time VVC encoder.