ScuDo

Scuola di Dottorato ∽ Doctoral School

WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation

Doctoral Program in Electronics and Communications Engineering (28thcycle)

High-Level Synthesis Based VLSI Architectures for Video Coding

By

Waqar Ahmad

******

Supervisor(s):

Prof. Guido Masera, Supervisor

Prof. Maurizio Martina, Co-Supervisor

Doctoral Examination Committee:

Prof. Matteo Cesana, Politecnico di Milano

Prof. Sergio Saponara, Università di Pisa

Prof. Enrico Magli, Politecnico di Torino

Politecnico di Torino

2017

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Waqar Ahmad
2017

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my loving parents, my wife–Sania and my lovely son–Mujtaba

Acknowledgements

I wish to express my sincere thanks to Prof. Guido Masera for his support and advice. I would like to express my very great appreciation for his support and advice, his visionary project ideas, and for providing such a great research environment. I would like to express my deep gratitude to Prof. Maurizio Martina, who served as PhD co-supervisor. His constant support, his enthusiastic encouragement and exciting research ideas and his constructive criticism have been an invaluable help for the success of this thesis. Also, I am particularly grateful for the technical and non-technical help given by my supervisor Prof. Guido Masera and Prof. Maurizio Martina. Furthermore, I want to thank all my colleagues from VLSI Lab. Polito for an excellent work environment and a great time. I would further like to thank the support and administrative team at VLSI Lab, who are doing a perfect job such that PhD students can focus on their research work. Finally, I wish to thank my wife Sania for her non-technical contributions to this work and for letting me follow my passion, my lovely son Mujtaba, my parents for their constant support in what I do, and my family and friends for reminding me of life outside my office.

Abstract

High Efficiency Video Coding (HEVC) is state-of-the-art video coding standard. Emerging applications like free-viewpoint video, 360degree video, augmented reality, 3D movies etc. require standardized extensions of HEVC. The standardized extensions of HEVC include HEVC Scalable Video Coding (SHVC), HEVC Multiview Video Coding (MV-HEVC), MV-HEVC+ Depth (3D-HEVC) and HEVC Screen Content Coding. 3D-HEVC is used for applications like view synthesis generation, free-viewpoint video. Coding and transmission of depth maps in 3D-HEVC is used for the virtual view synthesis by the algorithms like Depth Image Based Rendering (DIBR). As first step, we performed the profiling of the 3D-HEVC standard. Computational intensive parts of the standard are identified for the efficient hardware implementation. One of the computational intensive part of the 3D-HEVC, HEVC and H.264/AVC is the Interpolation Filtering used for Fractional Motion Estimation (FME). The hardware implementation of the interpolation filtering is carried out using High-Level Synthesis (HLS) tools. Xilinx Vivado Design Suite is used for the HLS implementation of the interpolation filters of HEVC and H.264/AVC. The complexity of the digital systems is greatly increased. High-Level Synthesis is the methodology which offers great benefits such as late architectural or functional changes without time consuming in rewriting of RTL-code, algorithms can be tested and evaluated early in the design cycle and development of accurate models against which the final hardware can be verified.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms / Abbreviations

$3D-HEVC$	3D High Efficiency Video Coding
AVC	Advanced Video Coding
CPU	Central Processing Unit
DSL	Digital Subscriber Line
DVD	Digital Versatile Disk
FF	Flip-Flop
$FPGA$	Field-Programmable Gate Array
FPS	Frames Per Second
HD	High Definition
HDL	Hardware Description Language
$HEVC$	High Efficiency Video Coding
HLL	High-Level Language
HLS	High-Level Synthesis
IEC	International Electrotechnical Commission
$ISDN$	Integrated Services Digital Network
ISO	International Organization for Standardization

$ITU-T$	International Telecommunication Union-Telecommunication
JTC	Joint Technical Committee
JVT	Joint Video Team
LAN	Local Area Network
LUT	Lookup Table
MB	Macro Block
MMS	Multimedia Messaging Services
$MPEG$	Moving Picture Experts Group
$PSTN$	Public Switched Telephone Network
RTL	Register Transfer Level
SD	Standard Definition
SoC	System on Chip
TV	Television
$UMTS$	Universal Mobile Telecommunications System
$VCEG$	Video Coding Experts Group
$VHDL$	VHSIC Hardware Description Language
$VHSIC$	Very High Speed Integrated Circuit
$VLSI$	Very Large Scale Integration
VoD	Video-on-Demand

Chapter 1

Introduction

This chapter starts with an introduction to the fundamentals of video coding through an historical perspective. Following this, the chapter surveys High-Level Synthesis (HLS) based video coding. Subsequently, we propose an alternative methodology for VLSI implementation of video coding algorithms and introduce its main components, i.e., the HLS based simulation, verification, optimization and synthesis. We conclude with an overview of the individual chapters, indicating the relevant contributions.

1.1 Introduction to Video Coding

The process of compressing and decompressing video is called video coding or video compression. Moving digital images are digitally compressed by video compression algorithms. There is a long list of the video coding applications, some applications of the video compression include TV, phones, laptops, cameras etc. Where there is a digital video content, there should be video compression behind that content. For the digital video large amount of the storage capacity is required if the video is in its original form i.e. uncompressed. As an example, uncompressed 1080p high definition (HD) video at 24 frames/second requires 806 GB of storage for a video of 1.5 hours duration with bit-rate requirement of 1.2 Gbits/second. That is why, for storage and transmission purposes of the digital video, video compression is a must, otherwise it will be impossible to store and process the uncompressed video contents for applications of today's era. Decompression of compressed video is required for displaying the video contents to the consumers.

Sending visual images to a remote location has captured the human imagination for more than a century Figure 1.1. The invention of television in 1926 by the Scotsman John Logie Baird [1] led to the realisation of this concept over analogue communication channels. Even analogue TV systems made use of compression or information reduction to fit higher resolution visual images into limited transmission bandwidths [2].

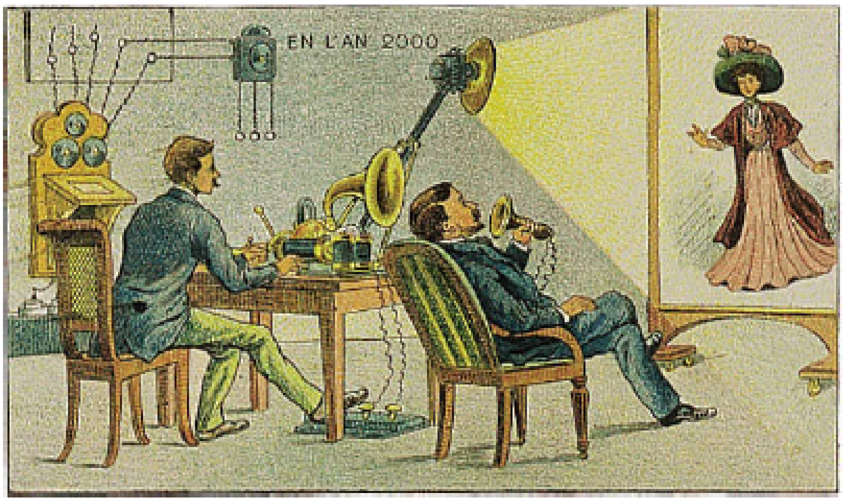


Fig. 1.1 "in the year 2000", postcard from 1910

The emergence of mass market digital video in the 1990s was made possible by compression techniques that had been developed during the preceding decades. Even though the earliest videophones [3] and consumer digital video formats were limited to very low resolution images (352x288 pixels or smaller), the amount of information required to store and transmit moving video was too great for the available transmission channels and storage media. Video coding or compression was an integral part of these early digital applications and it has remained central to each further development in video technology since 1990 [4].

By the early 1990s, many of the key concepts required for efficient video compression had been developed. During the 1970s, industry experts recognised that video compression had the potential to revolutionise the television industry. Efficient compression would make it possible to transmit many more digital channels in the bandwidth occupied by the older analogue TV channels.

Present-day video coding standards [5]–[6] and products share the following features:

1. Motion compensated prediction [7].
2. Subtraction of a motion compensated prediction for residual unit creation (e.g. a residual MB).
3. Block transform and quantization to form blocks of quantized coefficients.

1.2 High Level Synthesis Based Video Coding

Video compression technology can be seen in a variety of applications ranging from mobile phones to autonomous vehicles. Many video compression applications such as drones and autonomous vehicles requires real-time processing capability in order to communicate with the control unit for sending commands in real time. Besides real-time processing capability, it is crucial to keep the power consumption low in order to extend the battery life of not only mobile devices, but also drones and autonomous vehicles. Field Programmable Gate Arrays (FPGAs) are desired platforms that can provide high-performance and low-power solutions for real-time video processing. Increasing demands of multimedia applications and services has make up the need for embedded systems aiding ever-accelerating functionality and flexibility [8]. Evolution of video coding supporting new advanced coding tools and increased demand of multimedia contents make the embedded media processing systems difficult to design and implement, under shorter time-to-market restriction. Sate-of-the-art video coding standards i.e. HEVC [9] and H.264/AVC are good examples of complex multimedia system with low-power and typical performance embedded implementation requirements. Several works [10]–[11] has been proposed for the performance enhancement and complexity reduction of HEVC and H.264/AVC multimedia systems. As hardware designs typically are more time consuming than equivalent software designs. Due to difficult and time hungry process of manual RTL design, an alternative methodology for hardware implementations of complex system is High-Level Synthesis (HLS) based hardware implementation. Increased complexity of the digital systems [12], energy-efficient heterogeneous systems [13] for high-performance and shortening time-to-market, are the key factors for the popularity of the High-Level Synthesis (HLS) [14]. In HLS, hardware functionality is specified by using the software i.e. at a higher-level of abstraction. Moreover, field-programmable gate array (FPGA) design by HLS becomes interesting and has two fold advantage i.e. the hardware implementations in the target device can be easily replaced and refined at higher abstraction level.

Nowadays, heterogeneous-systems are being adopted as the energy-efficient, high-performance and high-throughput systems. The reason behind this is the impossibility of further clock frequency scaling. These systems consist mainly of two parts i.e., the application-specific integrated circuits (ASICs) [15] and the software processor [16]. Each part of the system is dedicated for a specific task. The design of these types of systems become very complex due to increase in the complexity of the systems. ASICs are the dedicated hardware components for the accelerated implementation of the computational complex parts for the system. As stated above, due to increase in the complexity of the systems, the design of these dedicated hardware also become complex and time-consuming. Hardware description languages (HDLs) [17] are used for the register transfer level (RTL) [18] implementation of these components. Cycle-by-cycle activity for RTL implementation of these components is specified, which is a low abstraction level. For such a low level of implementation, advanced expertise in hardware design are required, alongside being unmanageable to develop. The impact of these low-level implementation of complex systems increase the time-to-market by taking more design and development time.

High-level synthesis (HLS) and FPGAs in combination, is an intriguing solution to these problems of longer time-to-market and to realize these heterogeneous systems [19]. FPGAs are used for the configurable implementation of digital integrated circuits. Manufacturing cost is an important factor in the implementation of digital ICs. The use of FPGAs as reconfigurable hardware, help us the fast implementation and optimization by providing ability to reconfigure the integrated circuits, hence, removing the extra manufacturing cost. It allows the designer to re-implement modifications made to the design, by changing the HDL code description, re-synthesize and implement the design using same FPGA fabric by the help of implementation tools. Thus HLS based FPGA implementation of digital systems can be helpful in functional verification, possible hardware implementation and large design-space exploration of the systems. FPGA based implementation of user applications can be used an intermediate implementation before the ASICs and SoC implementation.

C, SystemC and C++ etc. are the High-level languages (HLLs) being used for the software programming and development. HLS tools take HLL as input and HDL description (circuit specification) is generated automatically. This automatically generated circuit specification performs the same functionality as software specification. Since, the benefits of HLS i.e. to have a new fast hardware implementation just by changing the code in software, help software engineers with very little requirement of the hardware expertise. The benefits of the HLS to hardware engineers are the fast, rapid and high-level abstraction implementation of complex systems design, thus increasing the possibility in design space exploration. For the fast and optimized implementation of the complex systems and designs having FPGAs as the implementation technology, HLS based implementation provides significant suitability in terms of alternative design-space explorations by facilitating implementations of the modifications made to the design [20].

The prominent developments in the applications of FPGA industry include the use of FPGAs in the acceleration of the Bing search by the Microsoft and the Altera acquisition by Intel [21]. These developments enhance the possibility of usability of FPGAs in computing platforms with the help of high-level design methodologies. Further recent applications of HLS include in the areas of machine learning, medical imaging, neural networks etc. The primary reason behind the application of HLS in above specified areas is energy and performance benefits [22].

As hardware designs typically are more time consuming than equivalent software designs, this thesis proposes a rapid prototyping flow for FPGA-based video processing system design. High-level synthesis tools translate a software design into hardware descriptive language, which can be used for configuring hardware devices such as FPGAs. The video processing algorithm design of this thesis takes advantage of a high-level synthesis tool from one of the major FPGA vendors, Xilinx. However, high-level synthesis tools are far from being perfect. Users still need embedded hardware knowledge and experience in order to accomplish a successful design. This thesis focuses on interpolation filter architecture design and implementation for high-performance video processing system designs using a high-level synthesis. The consequent design results in a frame processing speed of 41 QFHD, i.e. 3840x2160@41fps for H.264/AVC sub-pixel Luma interpolation, 46 QFHD for HEVC luma sub-pixel and 48 QFHD for HEVC chroma interpolation. This thesis shows the possibility of realizing a high-performance hardware specific application using software. By comparing our approach with the approaches in other works, the optimized interpolation filter architecture proves to offer better performance and lower resource usage over what other works could offer. Its reconfigurability also provides better adaptability of many video coding interpolation algorithms.

1.3 Problem Statement

In recent years, FPGA development has been moved towards higher abstraction levels. The move not only helps improve productivity, but also lowers the barrier for more algorithm designers to get access to the tempting FPGA platform. There is a wide selection of tools available in the market that can be used for high-level synthesis. Conventionally algorithm designers prefer using high-level languages such as C/C++ for algorithm developments, and Vivado HLS is one of the tools that is capable for synthesis C/C++ code into RTL for hardware implementation. Nevertheless, most high-level synthesis tools could not translate a high level implementation to a RTL implementation directly, and users must restructure the high level implementations in order to make them synthesizable and suitable for the specific hardware architecture. Therefore, it becomes important to adapt to the high-level synthesis tool and to discover approaches for achieving an efficient design with high performance and low resource usage. The high-level synthesis tool used in this work is Vivado HLS from Xilinx.

This thesis addresses the following issues:

1. How can engineers with limited FPGA experience quickly prototype an FPGA-based SoC design for high performance video processing system?

2. How productive is Vivado HLS? What changes need to be made in order for a software implementation to be synthesized to a hardware implementation?

3. How are the performance and area of video processing algorithms modelled by Vivado HLS compared to that of RTL modelling from related works?

4. How are the performance and power consumption of a FPGA-based video processing system compared to that of an Intel CPU based video processing system?

1.4 Contribution

This thesis work presents an FPGA-based video processing system rapid prototyping flow that aims to lower the boundary between software and hardware development. The rapid prototyping flow consists of two major parts: 1) the video processing system architecture design, and 2) the video processing algorithms design. By understanding the underlying architecture of Xilinx's Zynq platform, I can quickly assemble a video processing system on the block level with minimum RTL modifications. The development period can be reduced from months to weeks. In addition, since Vivado HLS does not provide a common structure for domain-specific algorithm designs, this thesis proposed HLS based hardware architecture for interpolation filters of video coding algorithm designs in Vivado HLS. Several optimizations are also done to the proposed interpolation filter architecture so that it not only improves the video processing rate, but also reduces the flip-flop utilization and saves the LUT utilization when comparing with similar works done in the literature. This work demonstrates the possibility of rapid prototyping of a computation-intensive video processing system with more than enough of the real-time processing performance.

1.5 Organization of the Thesis

This thesis is organized as the following: Chapter 2 discusses and compares the sate-of-the-art video coding standards i.e. High Efficiency Video Coding (HEVC), H.264/AVC and standardized extensions of HEVC. Also included in Chapter 2 are several related works that were done by others as well as some background information related to the coding tools that have been added in 3D-HEVC. Moreover, Chapter 3 describes the coding complexity analysis of 3D-HEVC. It identifies the computational intensive tools of 3D-HEVC encoder and decoder. Chapter 3 also discusses the class-wise coding and decoding time distribution of different classes (tools). In addition, Chapter 4 presents the High-level synthesis, available High-level synthesis tools and Xilinx Vivado Design Suite. Chapter 5 describes the HLS based implementation of interpolation filters of HEVC and H.264/AVC, which is one of the computational intensive part of the video coding algorithms. It includes the performance and resource utilization comparison between my work and other works.

Last but not least, Chapter 6 will conclude this thesis with discussion about the contributions, challenges and future work.

Chapter 2

State-of-the-art Video Coding Standards

For effective communication, standard define a common language to be used between different parties. The same holds equally valid for the video coding standards. The common language that the video encoding and decoding components use for communication and syntax of the bitstream, is defined by the video compression standards. For the video compression standards, it is very important to support efficient compression algorithms and allow efficient implementation of the encoder and decoder.

Coding efficiency optimization is the most important and primary goal of majority of the video coding standards. For specific video quality, coding efficiency can be defined as, minimization of the bit-rate required for representing the specified video quality. Other way around, for a specific bit-rate, the increase in the video quality can be termed as coding efficiency.

The goal of this chapter is to describe the state-of-the-art video coding standards being used i.e. High Efficiency Video Coding (HEVC) standard [5][23] and H.264/AVC [24][6] comparative to their major forerunner including H.262/MPEG-2 Video [25][26], H.263 [27] and MPEG-4 Visual [28].

2.1 History of The Standardization Process

An enabling technology for digital television systems worldwide was, the MPEG-2 video coding standard [25], which was an extension of MPEG-1. MPEG-2 was widely used for transmission of TV signals of High definition (HD) and Standard Definition (SD) over a variety of transmission media such as terrestrial emission, cable, satellite and for storage onto DVDs.

The popular growth of HDTV and its services increase the need for higher coding efficiency. Coding efficiency enhancement allows the transmission of high quality and higher number of video channels over already available digital media transmission infrastructures e.g. UMTS, xDSL, Cable Modem etc. These mediums allow less data rates as compared to the broadcast channels.

The evolution of video coding in applications of telecommunication include the development of H.261 [29], H.262 [25][26], H.263 [27], H.264/AVC [24][6] and H.265 (HEVC) [5] video coding standards. The prominent telecommunication applications are wireless mobile networks, ISDN, LAN and T1/E1. To maximize the coding efficiency, significant efforts dealing with the loss/error optimization, network types and formatting of the characteristic have been made. This evolution of the video coding standards expanded the capabilities like video shaping and broadened the application areas of the digital video.

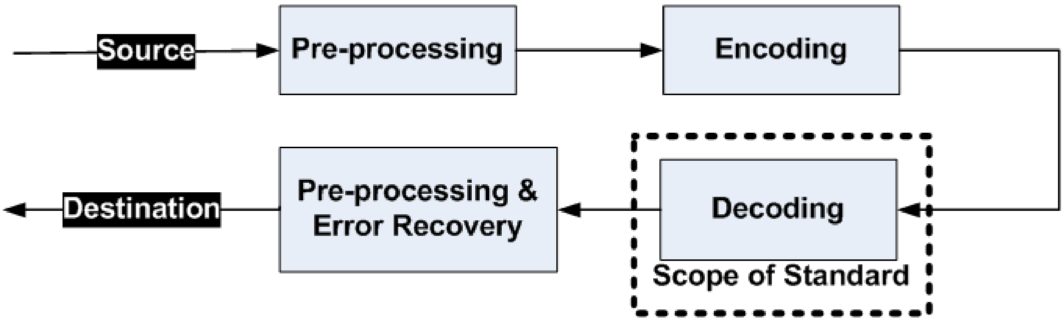


Fig. 2.1 Video coding standardization scope [6].

The scope of the video coding standard is shown in Fig.2.1. The transportation and storage media for video signal is not present in the scope of the video coding standard. The standardization of the decoder is central to video coding standard in all ISO/IEC and ITU-T standards. The standardization is about the syntax, bitstream structure and procedure for decoding the syntax elements. This makes for all the decoders to produce the same type of output when an encoded input bitstream conforming to the constraints of a specific standard is given. This limitation in standard's scope, allows the flexibility and freedom for optimized implementations e.g. time-to-market, quality of compression, cost of implementation etc. But there is no guarantee of reproduction quality as any crude coding technology can be conforming to the standard.

2.2 H.264/AVC Video Coding

H.264/AVC macroblock basic coding structure is shown in Fig. 2.2. By splitting the input video frame results in macroblocks, each macroblock is associated with a specific slice, slice consists of several macroblocks. As shown, processing on each macroblock of every slice is performed. Usually, one picture consists of various slices, in that case parallel processing is possible by processing more than one macroblocks in parallel.

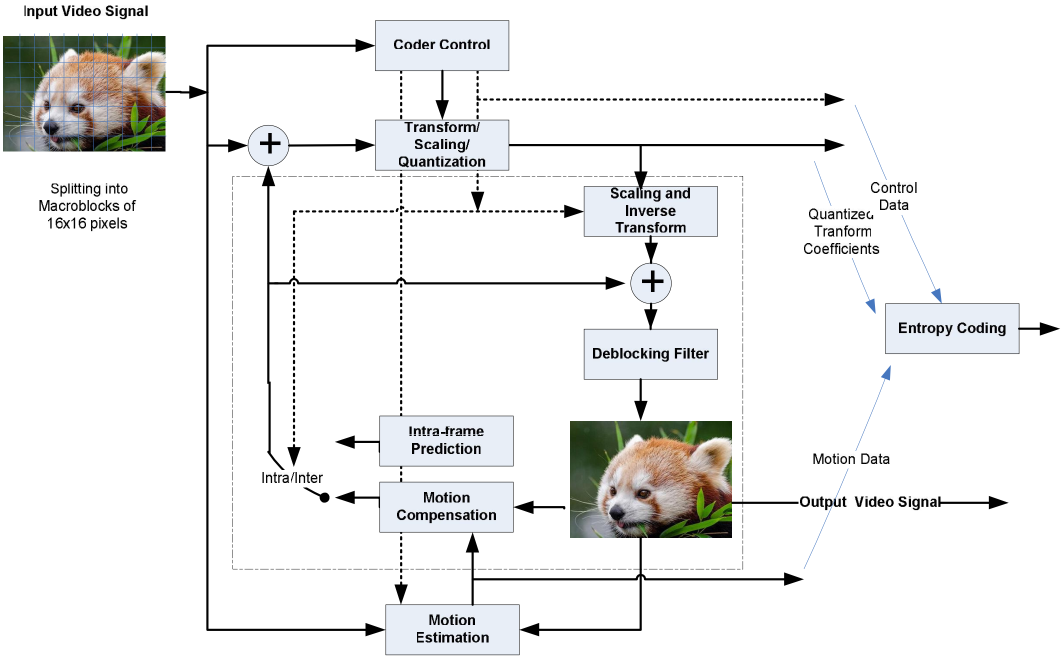


Fig. 2.2 H.264/AVC macroblock basic coding structure.

The prominent applications areas of H.264/AVC for which the technical solutions are designed include the following

- DSL, cable, terrestrial, satellite etc.
- Storage on DVD, optical and magnetic disks etc.
- Services over mobile networks, LAN, modems, Ethernet etc.
- Multimedia services like Video-on-demand, MMS over ISDN, DSL, wireless networks etc.

Some prominent features of the H.264/AVC are as follows [6].

- Supports motion compensation of small block sizes as 4 x 4 luma, hence, more flexible.
- More accurate motion vector by supporting quarter-sample motion compensation.
- Supports picture boundary extrapolation technique.
- For efficient coding supports enhanced reference picture selection.
- More flexible selection of pictures ordering for display and referencing purposes.
- Flexible picture referencing and representation methods.
- Supports weighted motion prediction.
- Supports "Skipped" motion inference in improved form and in addition to that supports "direct" motion inference method.
- Intra coding based on directional spatial prediction.
- Supports In-the-loop deblocking filtering.

For improvement in the coding efficiency, the following parts of the standard were also enhanced:

- Supports block size of 4x4 transform.
- Supports block transform in hierarchical manner.

- Supports 16-bit transform i.e. short, as comparative to 32-bit processing of the previous standards.
- Inverse transform is more efficient in terms of video content equality after decoding from all decoders.
- Includes CABAC (context-adaptive binary arithmetic coding) and CAVLC (context-adaptive variable-length coding) , more powerful and advanced entropy coding methods.

For the more robust and flexible operations, the new design features included in H.264/AVC standard are as follows:

- For the efficient and robust header information conveyance the Parameter set structure is provided.
- The logical data packet is used for every syntax structure, this is called NAL unit. This structure provides more flexibility in terms of customization for transmission of the video content over specific networks.
- Supports more flexible slice sizes.
- Supports Flexible macroblock ordering (FMO).
- Supports Arbitrary slice ordering (ASO), in real-time applications which can improve delay e.g., internet protocol networks.
- Supports Redundant pictures, which improves robustness to data losses.
- Supports Data Partitioning, allows the partitioning of the slice syntax up-to three different parts, for purpose of transmission, it depends on syntax elements categorization.
- Supports SP/SI synchronization/switching pictures, picture types specification enables synchronization of the decoder in decoding process of an ongoing video content stream produced by other decoders, hence improving the efficiency. This allows decoder switching between video content representations for different data rates, losses or errors recovery, enabling trick modes such as fast-reverse and fast-forward etc.

2.3 High Efficiency Video Coding (HEVC)

Growth in the popularity of HD video, increase in diversification of services, appearance of UHD and QFHD formats of video e.g. 4K or 8K resolution, demands the higher coding efficiency as compared to H.264/AVC's abilities. In addition to that, the demand for higher coding efficiency becomes more strong while considering the multiview or stereo applications of higher resolution. Furthermore, tablet PCs and mobile devices become the source of higher video contents consumption, application like video-on-demand needs efficient network infrastructure. Accumulatively, all these factors are imposing big challenges on the current networks. Mobile applications require higher resolutions and quality.

To address all existing H.264/AVC applications, need for a more efficient video standard becomes obvious. The most recent video project of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) standardization organizations is the High Efficiency Video Coding (HEVC). These organizations are collaborating as Joint Collaborative Team on Video Coding (JCT-VC) [23]. In 2013, HEVC's first edition was finalized in the form of an aligned draft published by both ISO/IEC and ITU-T. For extending the applications areas of HEVC standard more work was planned to support scalable, extended-range, 3D, multiview and stereo video coding. HEVC has been designed for this purpose and specifically to focus on two hot issues: increased applications of parallel architectures and higher resolution for video processing. Like all previous video compression standards of ITU-T and ISO/IEC, only the bitstream structure and syntax is standardized, and also the procedure for decoding.

2.3.1 HEVC Feature Highlights and Coding Design

HEVC has been designed to accomplish many goals, including integration of transport system, coding efficiency, resilience to data losses and architectures implementation using parallel processing.

Main features of the HEVC design are described briefly in the following paragraphs.

Video Coding Layer

Video coding layer utilize the hybrid approach for intraprediction, interprediciton and 2-D transform coding, the same approach was used in all previous video coding algorithms since H.261. Hybrid video encoder's block diagram is shown in Fig. 2.3, which could make a HEVC conformed bitstream.

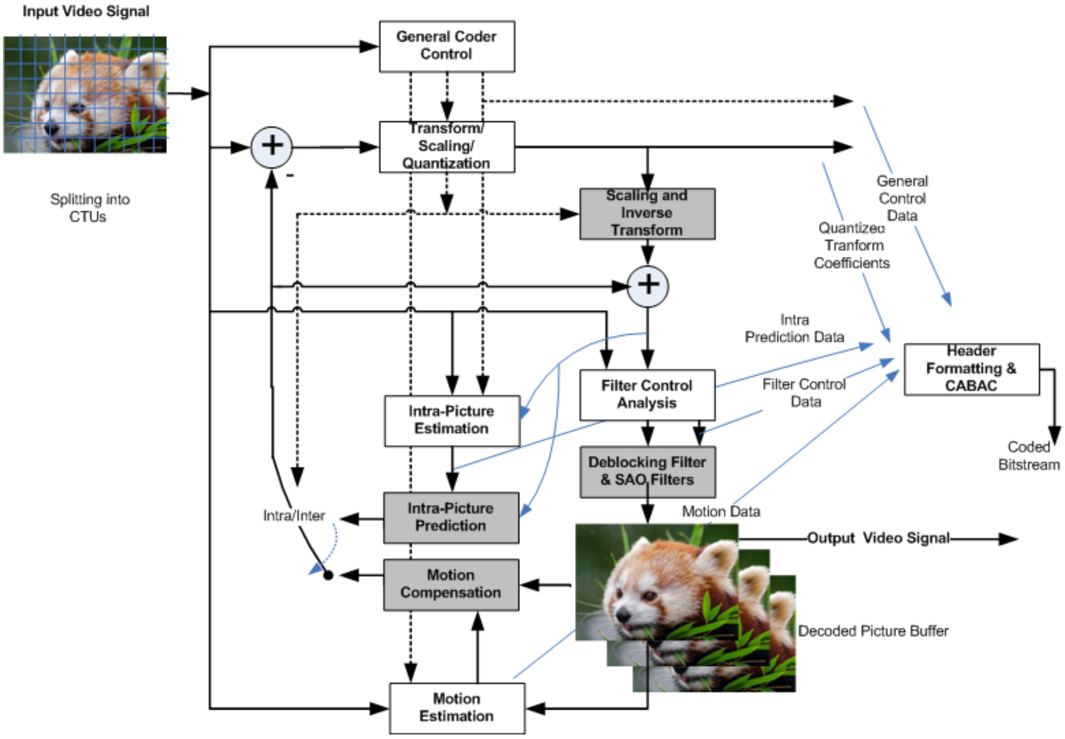


Fig. 2.3 HEVC video encoder (Light gray elements show decoder).

Highlighted features of HEVC are given in the following text. A more detailed version of these properties can be found in [9].

- ***Coding tree block (CTB) and Coding tree units (CTUs):*** One luma CTB, related chroma CTBs comprise the CTU. The size of the luma CTB can be LxL, where L= 16, 32, or 64 pixels. The larger the size the better the compression. CTBs are partitioned into smaller blocks of quadtree-like structure and signalling [30].
- ***Coding blocks (CBs) and Coding units (CUs) :*** One luma CB and two corresponding chroma CBs and the related syntax comprise a coding unit (CU). CUs are partitioned into prediction units (PUs) and transform units (TUs) tree.

- ***Prediction blocks (PBs) and Prediction units :*** Luma and chroma CBs can be further partition in size and predicted from luma and chroma PBs depending on the decision of the prediction-type. 64×64 down to 4×4 samples variable PB sizes are supported in HEVC.

- ***Transform blocks (TBs) and Transform units (TUs):*** Transforms blocks are used for the coding of prediction residual. Supported TB size are 32x32, 16x16 and 4x4.

- ***Motion vector signalling:*** In Advanced motion vector prediction (AMVP), most probable candidates are derived from reference picture and the adjacent PBs. For MV coding, a merge mode can be used. In merge mode, MVs are inherited from the spatially or temporally neighbouring PBs. Direct and improved skipped motion can also be used.

- ***Motion compensation:*** For the motion vectors (MVs), quarter-pixel precision is used. 7-tap and 8-tap filters are designed for the sub-pixel interpolation as compared to the H.264/AVC six-tap filters.

- ***Intrapicture prediction:*** 33 directional, DC (flat) and planar (surface fitting) prediction modes are supported in HEVC. The encoding of the selected prediction mode is performed based on neighbouring blocks previously decoded.

- ***Quantization control:*** HEVC supports uniform reconstruction quantization (URQ). For different transform block sizes the scaling matrices for quantization are used.

- ***Entropy coding:*** Context adaptive binary arithmetic coding (CABAC) method is used. The improvements made to the entropy coding method includes better coding performance, higher speed of throughput and reduction in the requirements of the context memory.

- ***In-loop deblocking filtering:*** More simplified deblocking filter in terms of filtering and decision-making, friendly in terms of parallel processing.

- ***Sample adaptive offset (SAO):*** Post deblocking in the interpicture prediction loop, a new type of non-linear amplitude mapping is used, for the better original signal reconstruction.

Modified Slice Structuring and Parallel Decoding Syntax

For slice data structure modification and parallel processing enhancement for the purpose of packetization, new features are added in the HEVC. In the context of a particular application, these features have specific benefits.

- ***Tiles:*** Partitioning a picture into rectangular area (Tiles) is supported in HEVC. Parallel processing capability can be increased by application of the concept of tiles. Tiles can be decoded independently with some common header information. Tiles support parallelism in terms of subpicture/picture i.e. coarse level of granularity.
- ***Wavefront parallel processing:*** WPP supports parallelism in terms of slice i.e. at a fine level of granularity. It gives better performance of compression as compared to tiles and removes the visual artefacts which may be present in case of tiles.
- ***Dependent slice segments:*** Dependent slice segment structure enables the fragmented packetization (separate NAL unit) of the data of a specific tile or wavefront entry. It improves the performance by reducing the latency. Low-level encoding may takes advantage of the dependent slice segments.

2.4 Standardized Extensions of HEVC

HEVC extensions can be divided into three types:

1. Range extensions
2. Scalability extensions
3. 3D video extensions

All of these extensions are briefly described in the following paragraphs:

2.4.1 Range Extensions

The structures of enhanced chroma sampling 4:2:2 and 4:4:4 and pixel bit depths more than 10 bits are supported in the range extensions of the HEVC. Range extensions are applicable to the areas of screen content coding, direct source content coding of the RGB, auxiliary pictures coding and lossless and high bit-rate coding. The draft range extensions can be found in [31].

2.4.2 Scalability Extensions

The coarse grain SNR and spatial scalability are possible through scalability extensions of the HEVC also termed as "SHVC". [32] provides the draft text of scalability extensions. SNR and spatial scalability in SHVC can combined with already available temporal scalability [33]–[34]. Resampling of the decoded reference layer picture is performed when spatial scalability is used. This resampling is performed by the use of upsampling filter defined specifically for the spatial scalability scenario.

2.4.3 3D Video Extensions

Depth for a visual scene can perceived by the multiview and 3D video formats in combination with the proper 3D display system. There are two type of the 3D displays available in the market:

1. **Stereoscopic displays:** Special glasses are required to perceive the depth of the view.

2. **Auto-stereoscopic displays:** No requirement of the glasses to perceive the depth of the scene, instead, view-dependent pixels are emitted. Auto-stereoscopic displays perform depth-image based rendering (DIBR). For DIBR systems, depth is part of the input coded bitstream. 3D formats in form of video plus depth is an important category of 3D formats.

Multiview HEVC

Mutiview HEVC is the most simple and straightforward architectural extension of HEVC, also termed as MV-HEVC, based on H.264/AVC MVC design principles [35], [36]. The draft text can be found in [37].

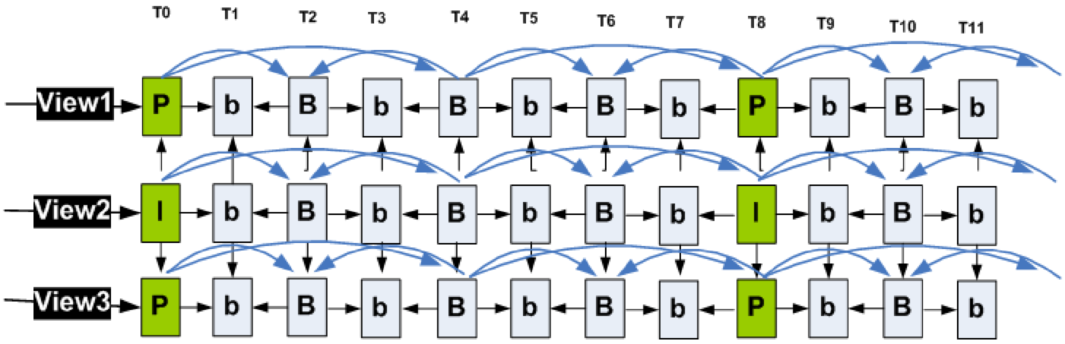


Fig. 2.4 3-view case: Prediction structure of Multiview HEVC.

The 3-view case prediction structure of the multiview is shown in Fig. 2.4. HEVC is capable of flexible management of the reference pictures. This capability of the HEVC enables the inter-view sample prediction.

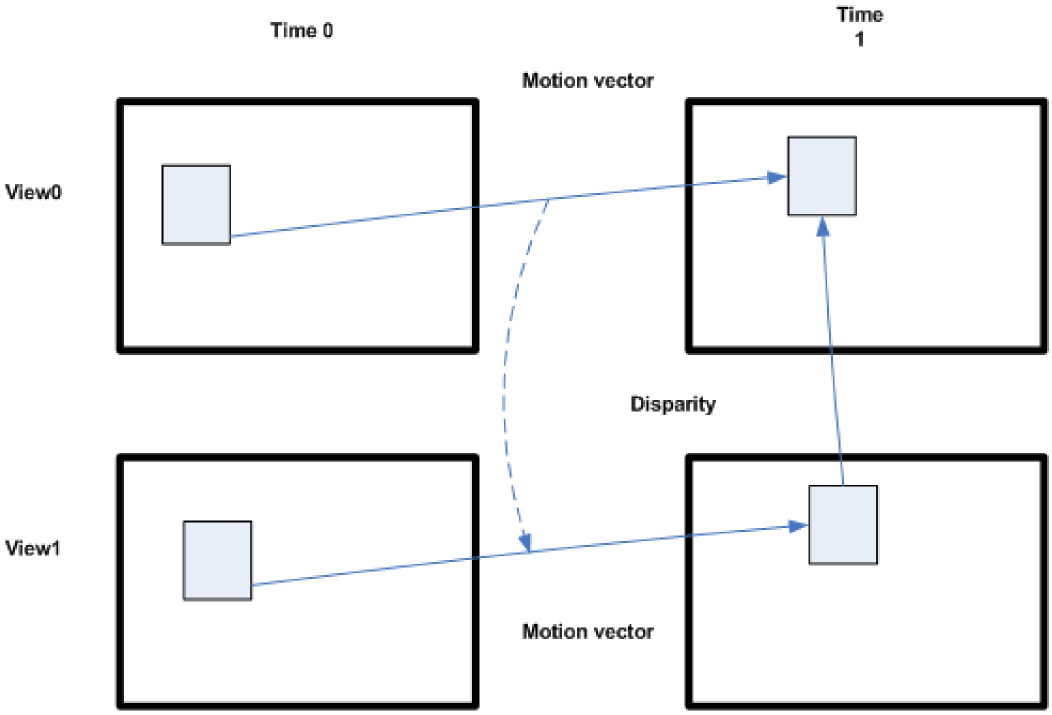


Fig. 2.5 HEVC Inter-view motion prediction.

Multiview HEVC With Modification in Block-Level Tools

The correlation between views exploited through residual and motion data. Block-level changes make possible the exploitation of this correlation as shown in 2.5

2.5 3D High Efficiency Video Coding (3D-HEVC)

3D-HEVC is the HEVC extension for which the working draft and the reference test model are specified in [38], [39]. The advanced coding tools for multiple views are included in this extension. [40] becomes the basis for the 3D-HEVC. Prominent 3D-HEVC tools are presented in the following paragraphs.

2.5.1 Neighbouring Block-Based Disparity Vector Derivation

Neighbouring block based disparity vector (NBDV) is the 3D-HEVC tool used for the identification of similar blocks in multiple different views. This tool's design is very similar to the merge mode and AMVP in HEVC. For inter-view pixel prediction of spatial and temporal neighbouring blocks, NBDV is used which make use of already available disparity vectors [41].

Fig. 2.6 shows the spatial neighbouring blocks used for the NBDV process, these are same blocks as in merge modes/AMVP of HEVC. The order of the block's access is also same as in merge mode: A_1, B_1 , B_0, A_0, and B_1.

2.5.2 Inter-View Motion Prediction

The merge mode modification by the addition of more candidates make the realization of the inter-view motion prediction. No modification to the AMVP is made. The new merge list has six candidates. The construction of the list is still same as in HEVC. Additional two candidates can be put into the list as described in the following text.

NBDV provides the index of the reference picture and motion vector of block found, as shown in Fig. 2.5. This is the first candidate inserted into the merge list. NBDV also provides the disparity vector and index of the reference inter-view picture. This is the second candidate inserted in merge list. Disparity vector insertion into the candidate list does not depend on existence of the inter-view candidate [42].

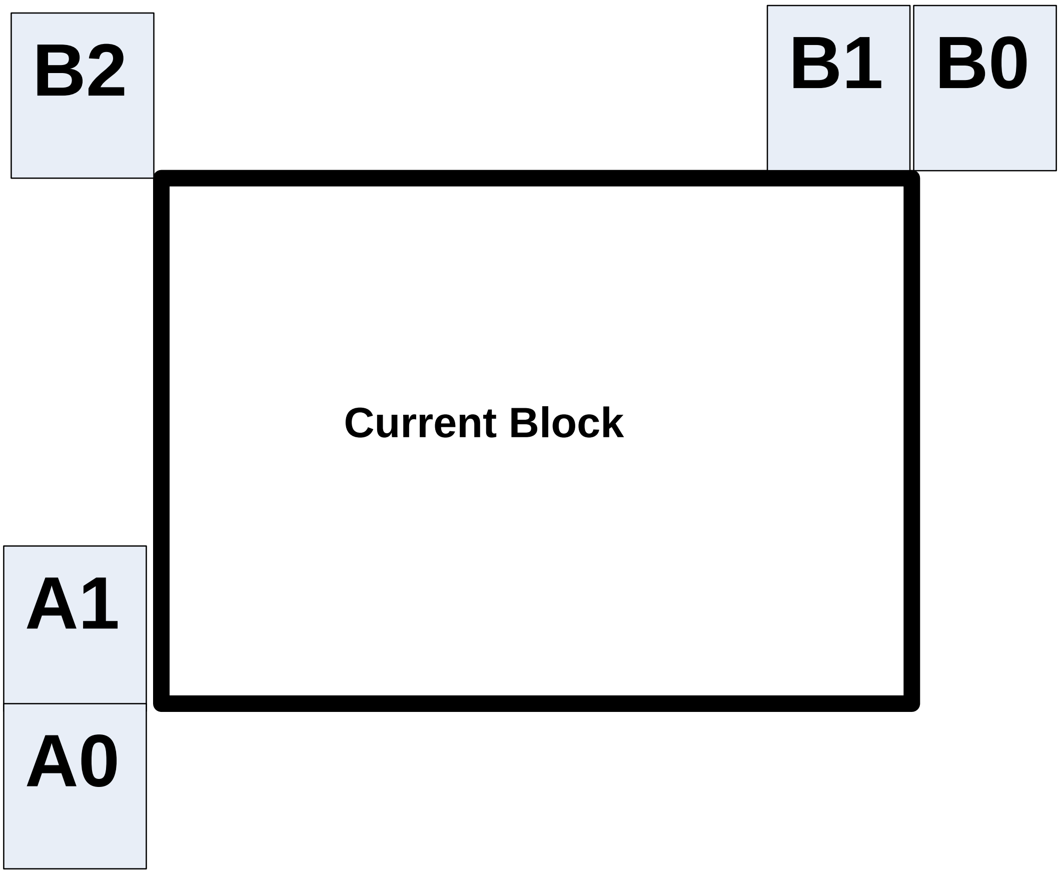


Fig. 2.6 Spatial Neighbouring blocks for NBDV.

Fig. 2.7 shows that the TMVP co-located block of view 1 at time 1 for current block, have a reference index 0 and disparity vector according to the current picture's temporal reference. That is why the TMVP candidate is usually regarded as unavailable. The candidate is regarded as available by changing the reference target index to 2 i.e. according to the inter-view reference picture.

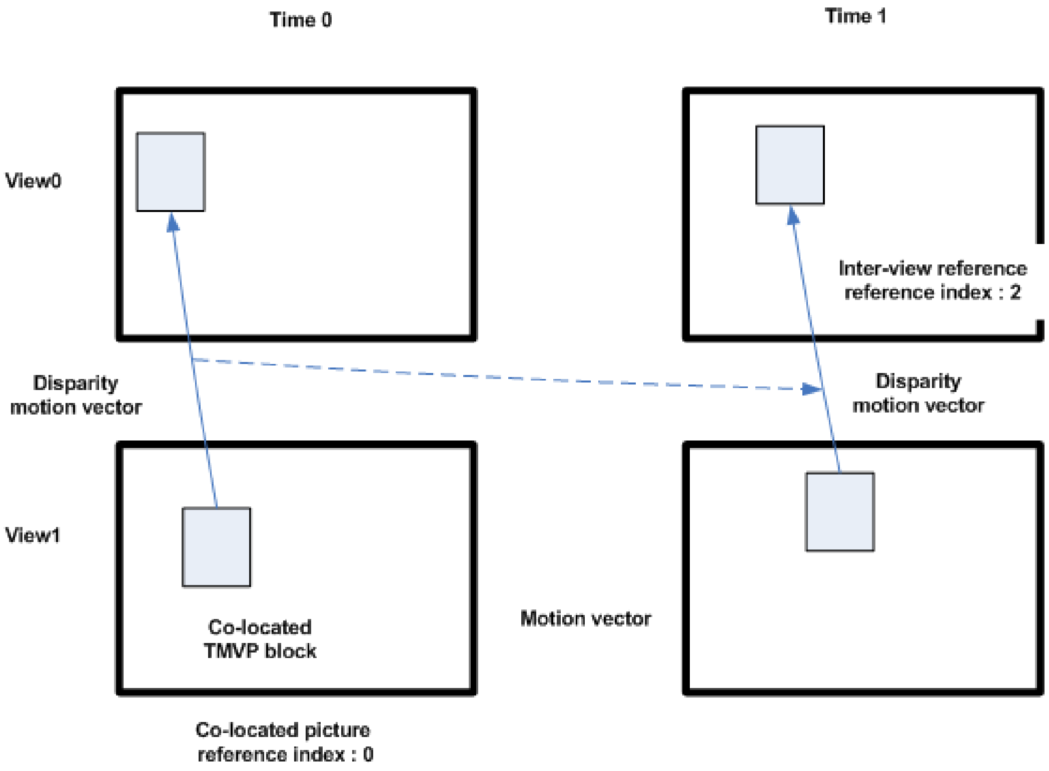


Fig. 2.7 3D-HEVC temporal motion prediction.

2.5.3 Inter-View Residual Prediction

In case of two-views residual signal motion-compensation, the advantage of the correlation is taken by the application of advanced residual prediction (ARP) [43].

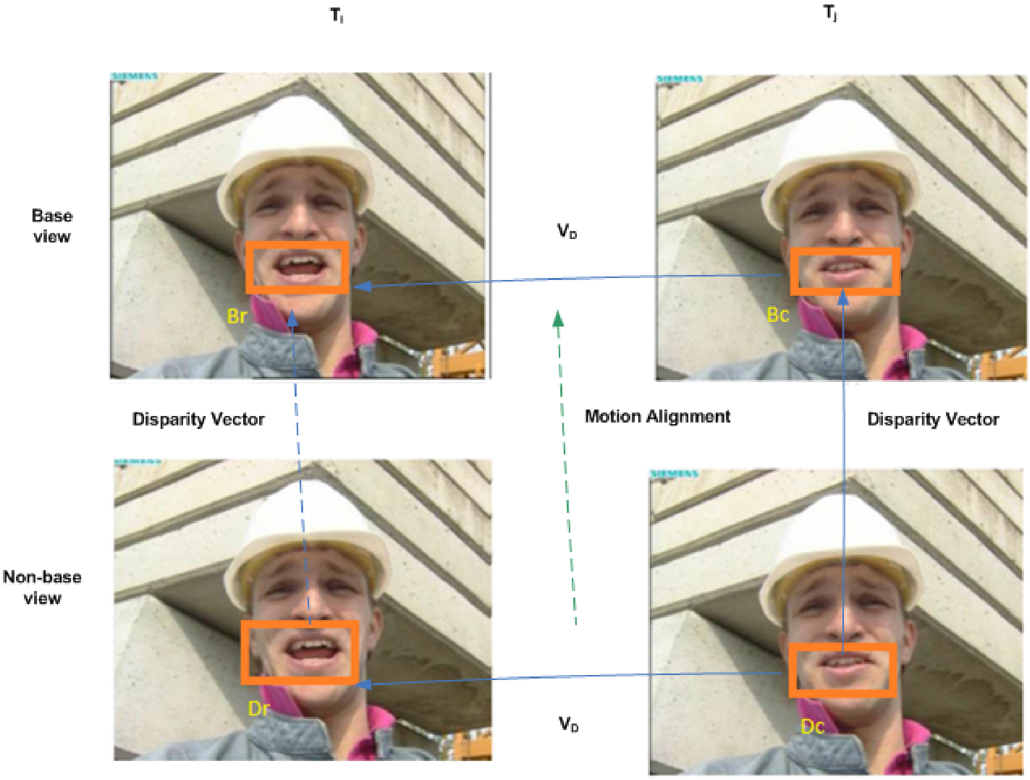


Fig. 2.8 3D-HEVC Temporal motion vector prediction.

In current non-base view i.e. for the block D_C, motion compensation is carried out using the V_D motion vector as shown in as shown in Fig. 2.8. The NBDV vector identifies the B_C inter-view block. Then, by the use of V_D, the motion compensation is performed by the base view reconstructed B_r and B_C. Addition of this predicted signal to the signal predicted by motion compensation of D_r is performed. The precision of the current block's residual signal is best as same V_D vector is used. This residual prediction can be can weighted by 1 or 0.5, with ARP enabled.

2.5.4 Illumination Compensation

The calibration of the cameras in lighting effects and colour transfer is very important. Otherwise, the prediction of the cameras recording the same scene may fail. For improvement in the coding efficiency of the blocks predicted through inter-view pictures, new coding tool named as illumination compensation is developed [44].

The disparity vector of the current PU is used for the identification of reference view neighbour sample as shown in Fig. 2.9.

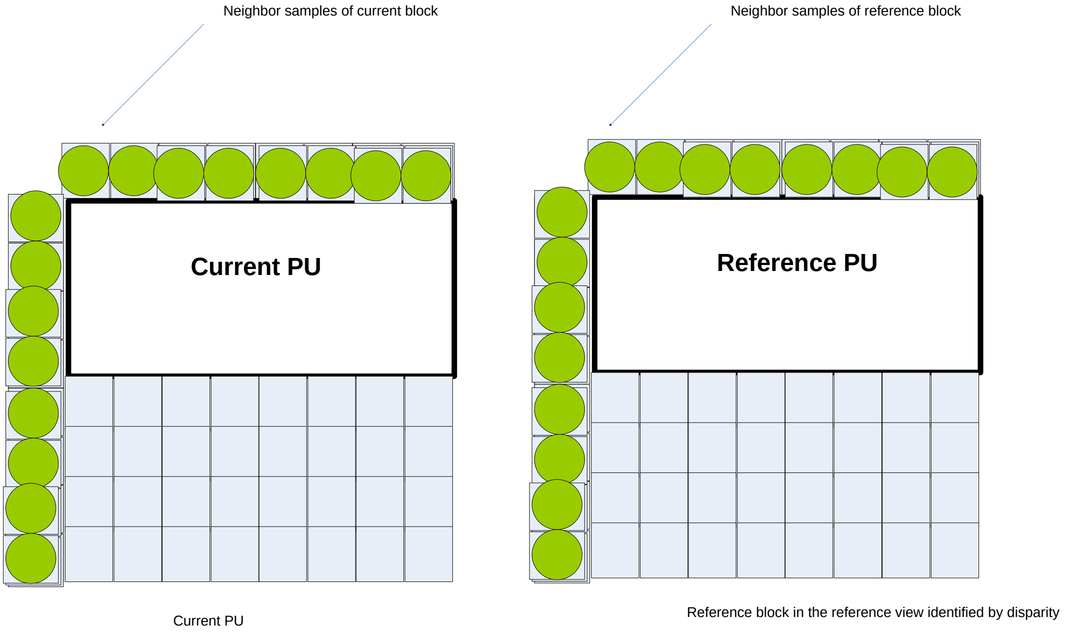

Neighbor samples of current block
Neighbor samples of reference block
Current PU
Reference PU
Current PU
Reference block in the reference view identified by disparity

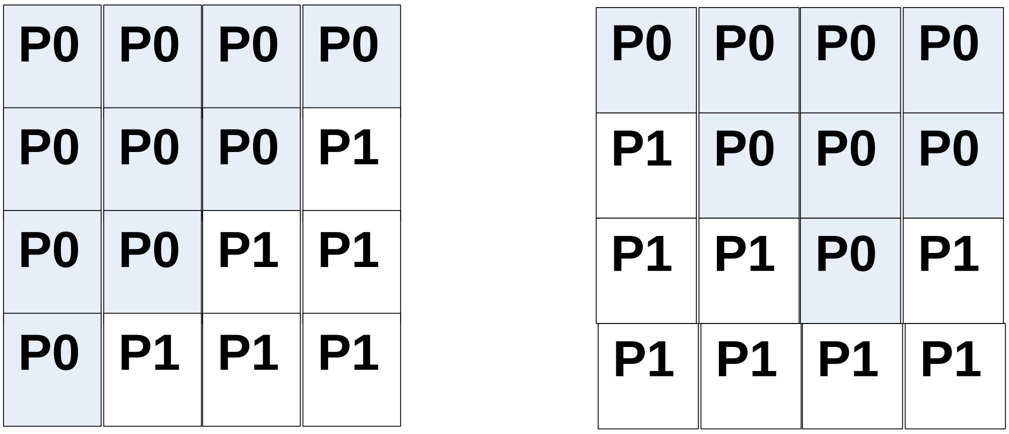
Fig. 2.9 Illumination Compensation.

2.5.5 Multiview HEVC With Depth

In 3D-HEVC, Depth maps are used in the investigation of compression formats such as video-plus-depth. The 3D data consists of multiple video and depth data components. For their efficient implementation, many new coding tools are added in 3D-HEVC for exploiting the correlation among video and depth data components. For coding of these types of formats, first video component is assumed as to be coded by 2D HEVC. This makes the codec compatible with the 2D video service. For the dependent video and depth maps, specific 3D tools are added in 3D-HEVC. Each block can be optimally coded by the application of appropriate tools from a set of 3D and 2D tools.

Partition-Based Depth Intra Coding

Coding tools specific to the depth for efficient depth information representation, are added in the 3D-HEVC design. These tools allow the non-rectangular partitioning of the depth blocks. Depth coding modes such as depth modelling modes (DMM) [45], simplified depth coding (SDC) [46] and region boundary chain coding (RBC) [47] are used for partition-based depth intra coding. Fig. 2.10 shows the division of depth PU as one or two parts. DC value is used for representing each part of the depth PU.



P0	P0	P0	P0
P0	P0	P0	P1
P0	P0	P1	P1
P0	P1	P1	P1

(a) Wedge-shaped pattern

P0	P0	P0	P0
P1	P0	P0	P0
P1	P1	P0	P1
P1	P1	P1	P1

(b) Boundary chain coding pattern

Fig. 2.10 Partitioning of depth PU.

Two types of depth partitioning are available in case of DMM. These are contour and wedge-shaped pattern. As shown in Fig 2.10(a), In case of the wedge-shaped pattern the depth PU is segmented by a straight line. Connected chain in a series fashion are used for segmenting the depth PU in case of RBC as shown in Fig. 2.10(b). Fig. 2.11 shows the partitioning of the depth PU based on contour pattern. As shown, these are irregular partitions with separate sub-regions.

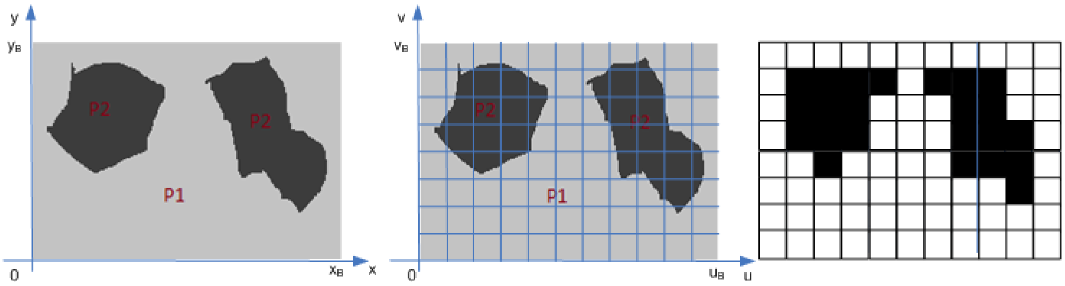


Fig. 2.11 Contour partition of a block.

Motion Parameter Inheritance

The motion parameters of the texture block can be used for the depth block. Merge list of current depth block is modified by the addition of one more candidate, making the inheritance of motion parameters of texture block for the corresponding depth. The co-located block of texture helps in the generation of the extra candidate [48].

View Synthesis Prediction (VSP)

For the reduction of the inter-view redundancy, the VSP approach is used. In this approach for texture view warping depth data information is used. By this method a current view predictor can be generated [49].

Chapter 3

Coding Complexity Analysis of 3D-HEVC

Recent advancements in video technology increased the interest in 3D Video. Video sequences are found in mobile, 3D cinema, internet and 3D television broadcast channels [50]. The quality of the video sequences is rapidly increasing due to the improvements in the video compression techniques and tools. Also, the improvements in the 3D video display technologies have led to an increased demand for 3D videos. Autostereoscopic displays are the future of displaying technologies in 3D Cinemas, 3D-TV and home entertainment. Video contents in resolutions are getting high definition and ultra-high definition for mobile and home applications respectively. 3D video features are already being integrated in most of the video processing devices including capturing, processing and display devices. The demand for compression of videos is also increased. The most recent and advanced standard for video compression is High Efficiency Video Coding (HEVC) [9]. After HEVC, focus is on extensions of the standard to support broad range of applications. 3D-HEVC is one of the extensions of High Efficiency Video Coding. Advanced coding algorithms are developed for 3D video coding. To analyse and assess the complexity, profiling of the reference software of 3D-HEVC is carried out using *gprof* and *gcc* compiler of the standard video sequences mentioned in the Common Test Conditions (CTC). Results based on profiling show that alongside motion estimation and interpolation filters, majority of the encoding time (18-26%) of total encoding time is consumed by the Renderer model. Thus, 3D-HEVC Renderer model is identified as one of the computational intensive part of the standard alongside motion estimation and interpolation filters. While some papers in the literature are available on the complexity evaluation of some tools of 3D-HEVC encoder/decoder, no results are currently available to specifically explore the complexity and hardware implementation analysis of renderer model of 3D-HEVC used for the View Synthesis Optimization (VSO). [51] presents time profiling of 3D-HTM 10.2 reference software, in which the complexity of texture and Depth Modelling Modes (DMMs) used for depth maps encoding, is given. Inter-prediction encoding time percentage for 3D-HTM 8.0 reference software is given in [52], no information is presented regarding the complexity analysis of rendering distortion estimation model for 3D-HEVC.

Renderer model is used for RDO of depth maps coding by estimation of synthesized view distortion. Depth maps are used for virtual view synthesis. Depth maps lie at the core of 3D video technologies. Distortion in depth maps coding effect the quality of intermediate virtual views generated during the process of DIBR. Because of these important observations and based on the profiling result, in Chapter 6, we have focused on the Renderer model. Identification of computational hotspots help both in decreasing the complexity and increasing the performance by developing the efficient tools and by implementing the accelerated software and hardware solutions for real time visualization of the 3D video coding standard.

3.1 3D-HEVC Tools

3D-HEVC basic structure is shown in Fig. 3.1. 3D-HEVC is an enhanced version of HEVC codec, for coding dependent views, the base view is coded using the HEVC codec.

Supplementary coding tools and techniques, as shown in Fig. 3.2, take into account the already coded data of other views, hence reducing the data redundancy.

3.1.1 Dependent View Coding

Dependent view coding in 3D-HEVC is performed by applying some supplementary coding methods in addition to the basic tools of independent view coding. Additional coding methods are used to decrease the data redundancy in the dependent view as described in the following paragraphs.

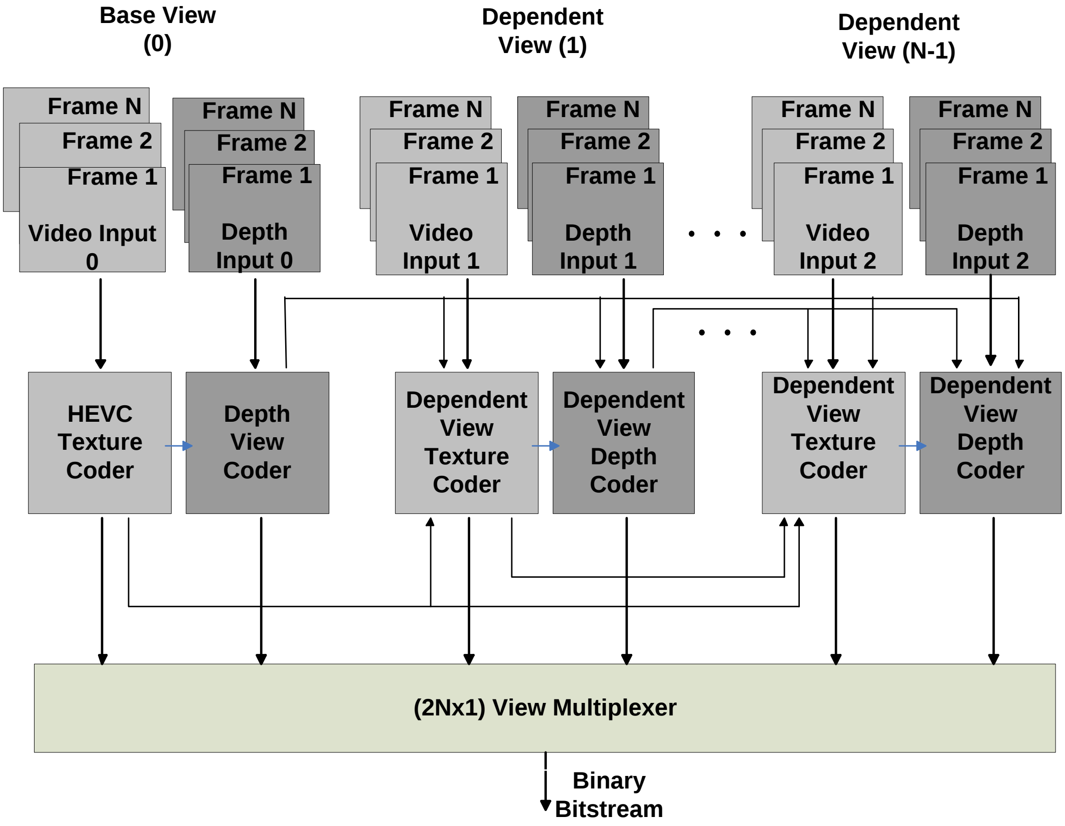


Fig. 3.1 Block Diagram of basic structure of 3D-HEVC.

Disparity-Compensated Prediction

Disparity Compensated Prediction (DCP) is used for the inter-view prediction of dependent views. The incorporation of DCP affects only the reference list construction procedure i.e. already coded pictures of other views and same access unit are added in the reference picture lists.

Inter-view Motion Prediction

Inter-view motion prediction is used for eliminating the data redundancy of the multiple views. The detailed description of Inter-view motion prediction is given in [53]. The motion information of current block of dependent view is obtained from corresponding block in the reference view.

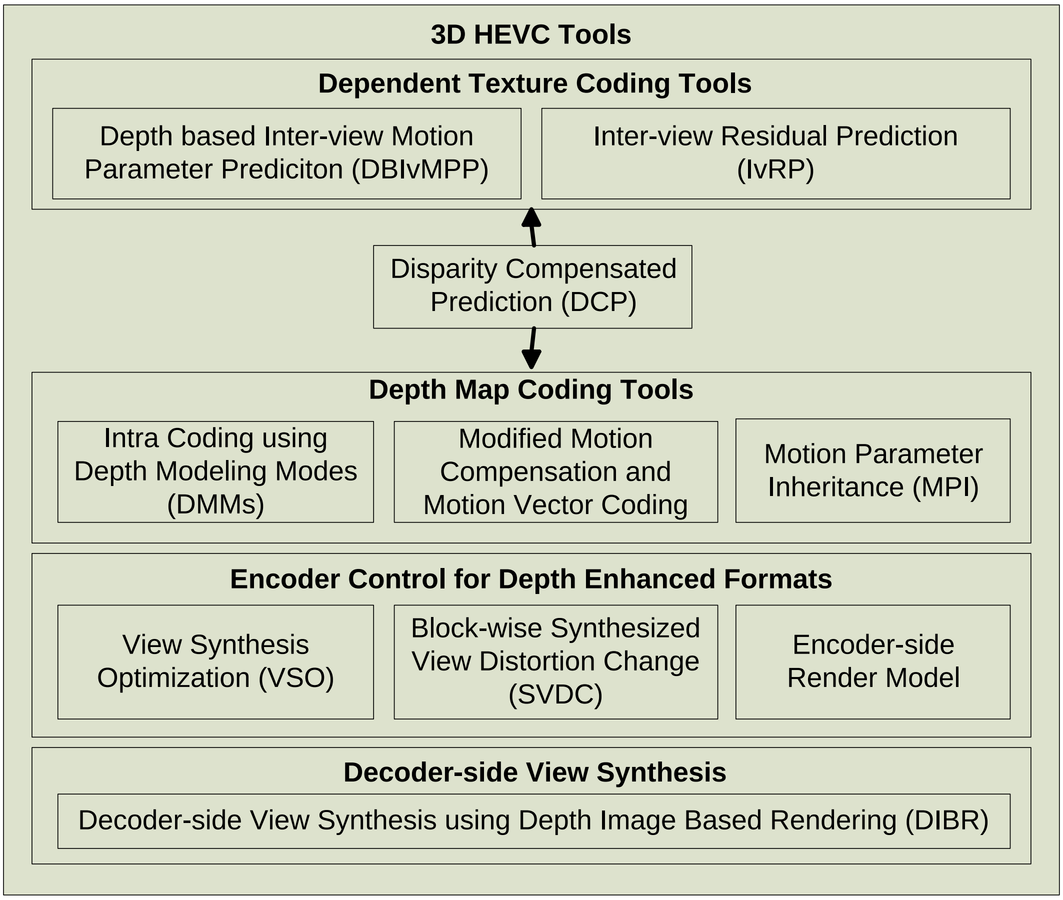

3D HEVC Tools
Dependent Texture Coding Tools
Depth based Inter-view Motion Parameter Prediciton (DBIvMPP)
Inter-view Residual Prediction (IvRP)
Disparity Compensated Prediction (DCP)
Depth Map Coding Tools
Intra Coding using Depth Modeling Modes (DMMs)
Modified Motion Compensation and Motion Vector Coding
Motion Parameter Inheritance (MPI)
Encoder Control for Depth Enhanced Formats
View Synthesis Optimization (VSO)
Block-wise Synthesized View Distortion Change (SVDC)
Encoder-side Render Model
Decoder-side View Synthesis
Decoder-side View Synthesis using Depth Image Based Rendering (DIBR)

Fig. 3.2 Block Level Representation of 3D Tools of HEVC.

Advanced Residual Prediction

In [54] the Advanced Residual Prediction (ARP) is described in detail. The correlation between the residual of already coded view and residual of current view also exist. To compensate this correlation advanced interview prediction is used.

3.1.2 Depth Maps Coding

Depth maps represent the distance of the objects in scene from the camera. Depth maps are used for view synthesis of intermediate views in multi-view generation systems. Depth maps consist of constant value regions with sharp edges. For depth maps intra-prediction, additional coding tools are used.

Depth Modelling Modes

In [55] depth modelling modes are introduced for coding of the depth maps intra-prediction. These tools divide the depth maps in two different non-rectangular regions of constant values for intra coding.

Motion Parameter Inheritance

Partitioning of a block to sub-blocks and motion information of the current block of depth map can be inherited from the corresponding block of texture as in [56].

3.1.3 Encoder Control

In 3D-HEVC, encoder mode is decided based on Lagrangian cost measure. Depth maps added for virtual view synthesis, the distortion measure for the depth maps can be observed only in synthesized views as described in [57]. Synthesized View Distortion Change (SVDC) is used for efficient estimation of distortion in rendered synthesized views due to distortion in depth maps coding.

3.2 Complexity Analysis

3D-HEVC is based on video plus depth format. Depth maps facilitate the synthesis of intermediate views on the decoder side for applications like 3D-TV, Free viewpoint TV etc. The compression errors of depth maps result in synthesis artefacts for the intermediate views rendered through Depth Image Based Rendering (DIBR) methods. To remove these coding artefacts in the virtual view synthesis process, the Synthesized View Distortion Computation (SVDC) models are included in 3D-HEVC. Encoding and decoding time Complexity analysis of 3D-HEVC standard is presented in this section. Profiling of the reference software of 3D-HEVC is carried out using *gprof* and *gcc* compiler of the standard video sequences mentioned in the Common Test Conditions (CTC). Results based on profiling show that (18-26%) of total encoding time is consumed by the Renderer model. Alongside other compute-intensive parts i.e. Motion Estimation (ME) and Interpolation Filtering, 3D-HEVC Renderer model is identified as one of the computational intensive part of the standard.

The 3D-HTM software provides the reference implementations of 3D-HEVC video encoder and decoder. Our aim is to identify computational hotspots of the standard. Implementation and complexity analysis of the standard may be assessed based on these computational hotspots. The detailed analysis of 3D-HEVC coding tools based on profiling results is given as in the following paragraphs.

3.2.1 Profiling of 3D-HTM Encoder

We have performed the profiling of latest available 3D-HTM Software Encoder Version 15.0 based on HM Version 16.6. GNU *gprof* is used for profiling. Computationally intensive parts are identified based on the profiling. CTC of 3DV Core Experiments are used for encoding the video sequences, used for profiling of the encoder [58]. Eight test video sequences (1024 x 768 pixels and 1920 x 1088 pixels) of 3DV Core Experiments are used for encoding at five different QP values (25, 30, 35, 40, 45). Two types of Encoder configuration i.e. Random Access (RA) and All Intra (AI), are used for the experiments. We have used the three-view case (C3) test scenario of the multiview/stereo video coding with depth data. Majority of the encoding time is spent in classes and functions shown in Table 3.1. Encoding times are obtained on an Intel Xeon-based (16 Core) Processor (E312xx clocked at 1.99 GHz) and using gcc 4.4.7.

Profiling Results comparison of 3D-HEVC and HEVC Encoder

Although in [51] and [52] partial profiling results of 3D-HEVC texture and depth maps are presented. We cannot directly compare our profiling results with results presented in [51] and [52] because our results are more detailed up-to class/function level. Table 3.1 shows the comparison between the profiling results of encoder of 3D-HEVC and HEVC [59] standards for Random Access (RA) and All Intra (AI) configurations, respectively. As shown in the Table 3.1, TComRdCost class consumes majority of the time spent in encoding i.e. about 31.3-35.4% and 9.8-38.8% in both configuration of 3D-HEVC and HEVC, respectively. Motion Estimation (ME), Inter view residual, Inter view motion prediction and other distortion operations takes place in TcomRdCost class. Operations like Sum of Absolute Difference (SAD), Hadmard transform (HAD) and Sum of Squared Error (SSE) for Rate Distortion Computation are performed. Depth maps estimation used in inter view motion prediction for the calculation of disparity vector derivation in dependent views is also calculated in this class. TRenSingleModelC class consumes about (26.8% and 18.3 %) of time. Process like VSO and SVDC estimation takes place in this class. Process of rendering is used for Virtual View Synthesis generation. In Random Access (RA) configuration, the time taken by TComInterpolationFilter class is about (19.3%) and (19.8%) , respectively, where the motion compensation Vertical and Horizontal Filtering (VHF) occurs. Interpolation filtering is used, whenever the inter-view residual prediction, de-blocking and View Synthesis prediction is applied.

Table 3.1 Class-wise time distribution 3D-HEVC vs HEVC Encoder.

Function / Class	3D-HEVC		HEVC [59]	
	AI%	RA%	AI%	RA%
TComRdCost	35.4	31.3	9.8	38.8
TRenSingleModelC	26.8	18.3	Nil	Nil
TComInterpolationFilter	0.0	19.3	0.0	19.8
TComTrQuant	9.0	10.0	24.4	10.7
TEncSearch	8.7	3.6	11.8	7.4
TComPrediction	5.0	0.88	10.0	1.1
partialButterfly*	2.3	4.1	8.7	4.0
TEncSbac	2.9	1.8	8.4	3.5
TRenModel	0.4	2.3	Nil	Nil
TComDataCU	2.2	1.0	5.8	2.7
TComPattern	2.2	0.2	6.6	0.4
TComYuv	0.2	1.8	0.1	1.7
TEncEntropy	Nil	Nil	1.2	0.6
TEncBinCABAC*	Nil	Nil	2.2	0.9
memcpy/memset	Nil	Nil	11.0	7.1
Total percentage of time	95.1%	95.3%	100%	98.7%

In 3D-HEVC and HEVC, TComTrQuant class accounts for about (9% and 10%) and (24.4% and 10.7%) of total encoding time, respectively. In TComTrQuant the process of Rate-Distortion Optimized Quantization (RDOQ) occurs. As the name of class shows, in TComTrQuant, the process of rate and distortion optimized transform and quantization takes place. TEncSearch accounts for about (8.7 % and 3.6 %) and (11.8 % and 7.4 %) of the time in both configurations of HEVC encoders, respectively. In TEncSearch , the encoder searches for the cost and rate-distortion computation of modes for inter, intra, depth intra for DMM, for motion estimation processes and Advanced Motion Vector Prediction (AMVP) of HEVC based standards. Similarly for intra prediction classes like TComPrediction and TComPattern contribute about (2% to 7%) to the total encoding time in both configuration of 3D-HEVC. Actual optimized Transform takes place in partialButterfly* and contribute about (2.3 % and 4.1 %) and (2.3 % and 4.1 %) in both configurations of the standards. Other classes like TEncSbac,TComDataCU and TComYuv contribute about (1%-3%) to total encoding time in both configurations.

3.2.2 Profiling of 3D-HTM Decoder

Profiling of 3D-HTM Software Decoder Version 15.0 based on HM Version 16.6 is carried out using GNU *gprof*. Computationally intensive critical parts of decoder are identified based on the profiling information.

Profiling Results comparison of 3D-HEVC and HEVC Decoder

Table 3.2 shows the decoding time distribution of 3D-HEVC and HEVC Decoder. Classes contributing significantly in terms of time consumption, in the decoding process, are shown. In all intra configuration more than quarter of total time is spent in TComInterpolationFilter. In the process of motion compensation, interpolation filtering is used. TComCUMvField, TComLoopFilter, TComDataCU classes also account for most of the decoding time. In these classes the processes internal to CU, advance motion vector prediction and filtering takes place. TComYuv is a general YUV buffer class, it manages the memory related functionalities of decoder. In random access configuration, partialButterflyInverse, TComPattern, TDecCu and TComLoopFilter classes are computationally intensive classes in the decoding process. In these classes processes related to inverse transform, functions related to coding unit, intra prediction and loop filtering takes place. In HEVC and 3D-HEVC, the computational complexity of classes varies from one standard to the other, as observed from the profiling results.

Table 3.2 Class-wise time distribution 3D-HEVC vs HEVC Decoder.

Function / Class	3D-HEVC		HEVC [59]	
	AI%	RA%	AI%	RA%
TComInterpolationFilter	0.0	26.96	0.0	24.8
TComCUMvField	7.39	16.03	Nil	Nil
TDecCu	15.97	3.69	7.2	2.6
partialButterflyInverse	15.83	1.80	15.9	7.6
TComYuv	0.83	14.42	0.5	8.2
TComDataCU	7.54	13.89	7.5	7.1
TComLoopFilter	13.73	8.94	12.9	12.4
TComPattern	10.19	0.0	9.4	2.6
TComTrQuant	8.07	2.74	8.7	4.2
TComPrediction	5.22	2.06	5.1	2.3
TDecSbac	3.84	0.0	6.2	2.8
TDecBinCABAC	2.93	.42	5.3	2.3
TComSampleAdaptiveOffset	2.64	1.04	3.8	2.4
TComPicYuv	0.0	2.3	Nil	Nil
writeplane	1.06	1.68	Nil	Nil
TDecEntropy	1.23	.71	1.4	1.0
memcpy/memset	Nil	Nil	6.2	10.1
Total percentage of time	96.47%	96.68%	90.1%	90.4%

3.3 Identified Computational Complex Tools

Fig. 3.3 shows the identification and mapping of computationally intensive parts of the 3D-HTM standard. The identification of these parts is carried out by mapping the profiling results of C++ HTM encoder and decoder classes to 3D-HEVC High level encoder coding tools. From the profiling results, it is identified that the major part of the encoding time of 3D-HEVC is consumed in motion estimation including interview motion prediction, encoder control regions consisting of the VSO, SVDC by the use of rendering method and interpolation filters, as shown in Fig. 3.3. Identified computational intensive parts of 3D-HEVC standard are listed as follows:

1. Motion Estimation (ME)

2. Synthesized View Distortion Change (SVDC)

3. Interpolation Filters

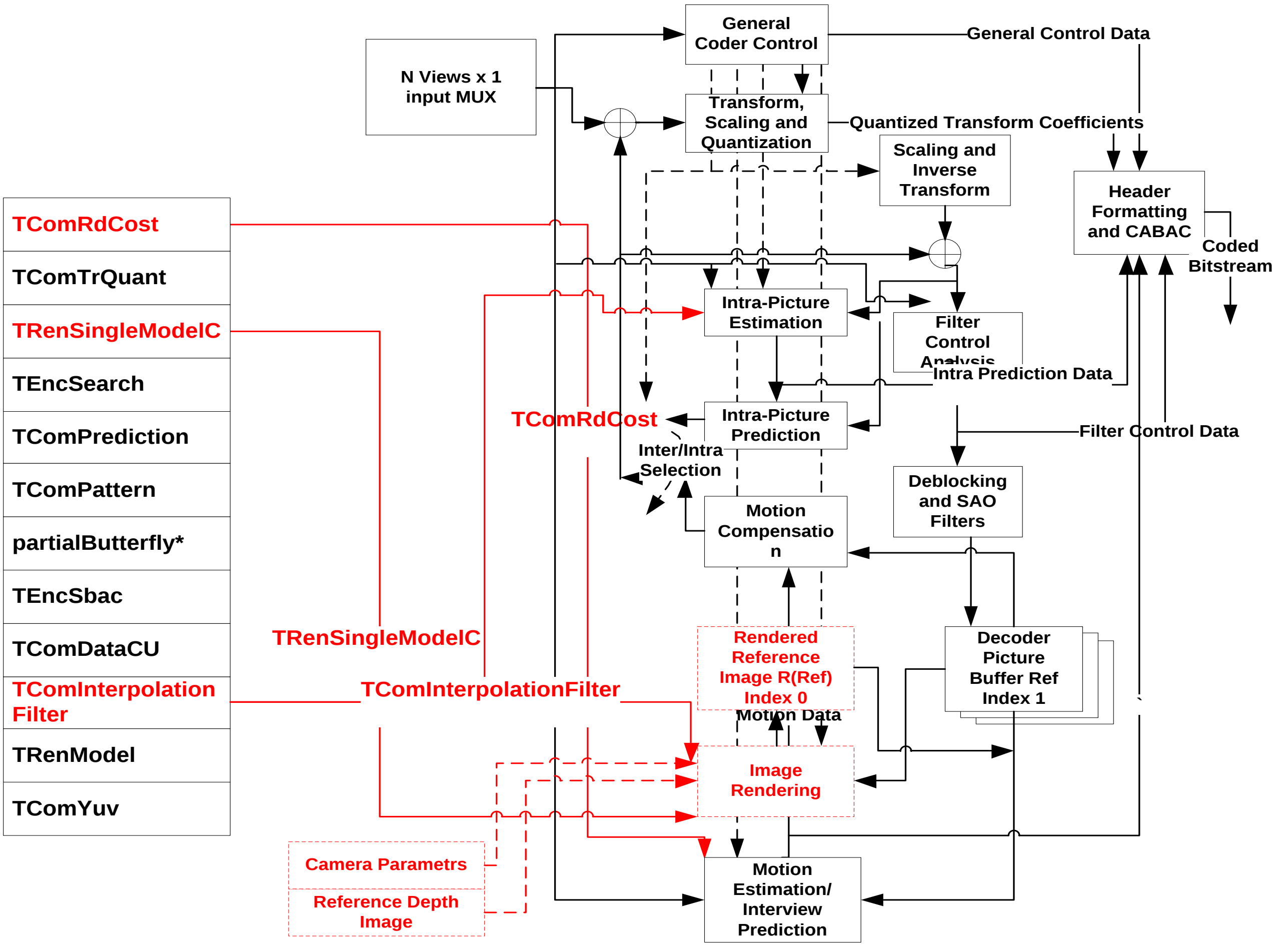


Fig. 3.3 Identification and mapping of Computationally Complex parts of 3D-HEVC.

Chapter 4

High-Level Synthesis

In this chapter, an analysis of HLS techniques, HLS tools, current HLS research topics are presented. For the automatic design of customized application-specific hardware accelerators, academia and industry are working together. Three academic tools considered are Delft workbench automated reconfigurable VHDL generator (DWARV) [60], BAMBU [61], and LEGUP [62] alongside many other commercial available tools. Many research challenges are still open in HLS domain.

4.1 What is High-Level Synthesis?

Nowadays, heterogeneous-systems are being adopted as the energy-efficient, high-performance and high-throughput systems. The reason behind this is the impossibility of the further scaling of the clock frequency. These systems consist mainly of two parts i.e., the application-specific integrated circuits (ASICs) and the software processor [16]. Each part of the system is dedicated for a specific task. The design of these types of systems become very complex due to increase in the complexity of systems. ASICs are the dedicated hardware components for the accelerated implementation of the computational complex parts for the system. As stated above, due to increase in the complexity of systems, the design of these dedicated hardware also become complex and time-consuming. Hardware Description Languages (HDLs) are used for the register transfer level (RTL) implementation of these components. Cycle-by-cycle activity for RTL implementation of these components is specified, which is a low abstraction level. For the such a low level of implementation, advanced expertise in the hardware design are required, alongside being unmanageable to develop. The impact of these low-level implementation of complex systems increase the time-to-market by taking more design and development time.

High-level synthesis (HLS) and FPGAs in combination, is an intriguing solution to these problems of longer time-to-market and to realize these heterogeneous systems [19]. FPGAs are used for the configurable implementation of digital integrated circuits. Manufacturing cost is an important factor in the implementation of digital ICs. The use of FPGAs as reconfigurable hardware, help us the fast implementation and optimization by providing the ability to reconfigure the integrated circuits, hence, removing the extra manufacturing cost. It allows the designer to re-implement modifications made to the design, by changing the HDL code description, re-synthesize and implement the design using the same FPGA fabric by the help of implementation tools.

C, SystemC and C++ etc. are High-level languages (HLLs) being used for the software programming and development. HLS tools take HLL as input and then HDL description (circuit specification) is generated automatically. This automatically generated circuit specification perform the same functionality as the software specification. Since, the benefits of HLS i.e. to have a new fast hardware implementation just by changing the code in software, help software engineers with very little requirement of the hardware expertise needed. The benefits of the HLS to hardware engineers include are the fast, rapid and high-level abstraction implementation of complex systems design, thus increasing the possibility in design space exploration. For the fast and optimized implementation of the complex systems and designs having FPGAs as the implementation technology, HLS based implementation provides significant suitability in terms of alternative design-space explorations by facilitating implementations of the modifications made to the design [20].

The prominent developments in the applications of the FPGA industry includes the use of FPGAs in the acceleration of the Bing search by the Microsoft and the Altera acquisition by Intel [21]. These developments enhance the possibility of usability of FPGAs in computing platforms with the help of the high-level design methodologies. Further recent applications of HLS include in the areas of machine learning, medical imaging, neural networks etc. The primary reason behind the application of HLS in above specified areas is due to the energy and performance benefits [22].

4.2 Overview of High-Level Synthesis Tools

HLS tools overview is presented in this section. As shown in Figure 4.1, HLS tools are presented by classifying the design input language. Two classes of the tools are made based on input languages. First category of tools accept general-purpose languages (GPLs) and the second category of the tools accept domain-specific languages (DSLs) as input. Further splitting of the DSLs tools is made on the basis of tools invented for GPL-based dialects and for a specific tool-flow. The tools are categorized, in each category red, blue and green fonts are used for the tools. Where red shows the tool is obsolete, blue represents N/A i.e. no information about the usability status of the tool and green shows the tool is still in use. The figure legends show the application areas of the tool. The use of SystemC or DSLs as input language increase the chances of tools adoption by the software developers.

In the following paragraphs we presented available commercial and academic HLS tools with brief description. Information regarding the target application domain, automatic generation of test bench and support for fixed and floating arithmetic can found in [63].

4.2.1 Academic HLS Tools

- **DWARV:** Developed by ACE, this tool is based on commercial infrastructure of CoSy compiler [60], it has robust and modular back-end.
- **BAMBU:** Developed by Politecnico di Milano, has the ability to produce Pareto-optimal solutions to trade-off resource and latency requirements [61].
- **LEGUP:** Developed by University of Toronto, this tool is based on virtual machine compiler framework (LLVM) [62]. Supports OpenMP and Pthreads i.e. parallel hardware are synthesized automatically from parallel software threads.

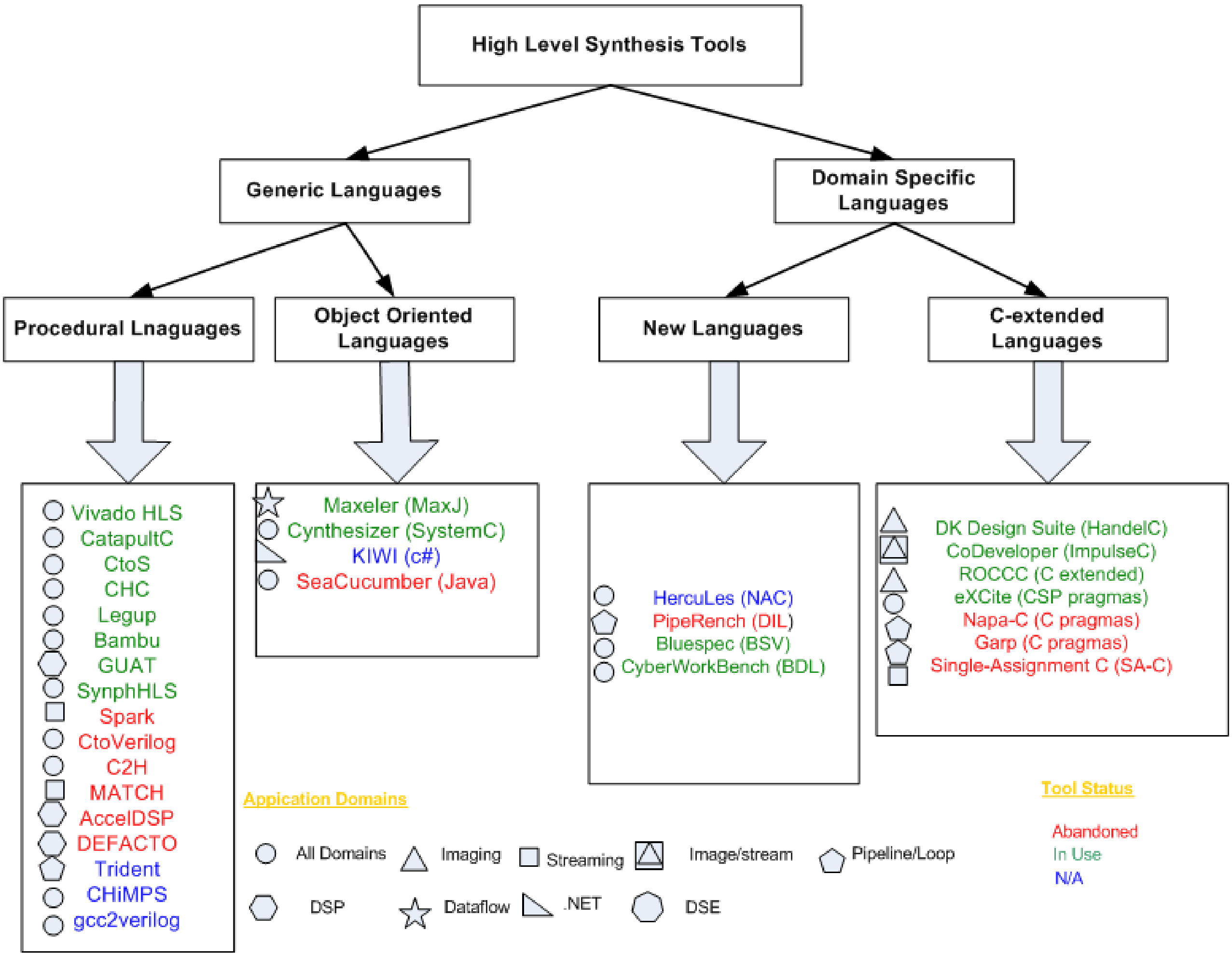


Fig. 4.1 HLS Tools Classification.

4.2.2 Other HLS Tools

- **CyberWorkBench:** Specifically developed for system-level design accepting behavioural description language (BDL) as input [64].
- **Bluespec Compiler (BSC):** Design language is Bluespec System Verilog (BSV), special expertise are required for the designers to use BSV [65].
- **PipeRench:** Originally developed for streaming applications for producing reconfigurable pipelines [66].
- **HercuLeS:** Based on a typed-assembly language accessible through GCC Gimple and used for only FPGA targeted applications [67].

- **CoDeveloper:** Developed by Impulse accelerated technologies, C-language based Impulse-C, only streaming and image processing applications are supported [68].
- **DK Design Suite:** Design language is Handel-C, an extended hardware focussed version of C language [69].
- **Single-Assignment C (SA-C):** Design language is based on C-language, only one time setting of variables is supported [70].
- **Garp:** Main aim of this project was the loops acceleration of general-purpose (GP) software services [71].
- **Napa-C:** Developed at Stanford University, this was the first project which considers systems containing configurable logic and microprocessors compilation based on high-level synthesis [72].
- **eXCite:** Supports manual insertion of communication channels between hardware and software [73].
- **ROCCC:** Mainly developed for parallel implementation of computational dense heavy applications [74]
- **Catapult-C:** HLS tool initially developed for ASICs but now it is used for both ASICs and FPGA [75].
- **C-to-Silicon (CtoS):** Developed by Cadence used for both dataflow and control applications. SystemC is the input language [76].
- **SPARK:** Targets image processing and multimedia applications, generated VHDL and can be implemented on FPGA and ASICs [77].
- **C to Hardware Compiler:** Application specific processor core based hardware design with manual verification [78].
- **GAUT:** This can produce communication, memory and accelerator hardware units with automatic testbench generation [79].
- **Trident:** Produce VHDL based hardware accelerators for floating point applications [80].

- **C2H:** Technology dependent tool targeting Altera soft processor and Avalon bus based hardware accelerator units [81].
- **Synphony C:** HLS tool Developed by Synopsys for DSP hardware design supports loop pipelining and loop unrolling [82].
- **MATCH:** Implementation of image and signal processing heterogeneous systems based on MATLAB code [83].
- **CHiMPS compiler:** Targets high performance applications by optimized implementation of FPGA memories [84].
- **DEFACTO:** Supports software/hardware co-design for computational intensive applications [85].
- **MaxCompiler:** Accepts java-based MaxJ as input language and produces Maxeler hardware based data-flow specific hardware engines [86].
- **Kiwi:** Generates verilog based FPGA co-processor from C# code [87].
- **Sea cucumber:** Java-based compiler produces electronic design interchange format netlists [88].
- **Cynthesizer:** Supports formal verification between gates and RTL, FP operations and power analysis [89].
- **Vivado HLS:** AutoPilot [90] initially developed by AutoESL, later Xilinx acquired AutoPilot in 2011 and it becomes Vivado HLS [91]. Xilinx HLS is based on LLVM and released in early 2013. This improved product includes in it, a complete environment for the design with rich characteristics for generation of fine-tune HDL from HLL. Accepting C++, C and SystemC as the input and generating hardware modules in Verilog, VHDL and SystemsC. At the time of compilation it gives the possibility of applying various optimizations such as loop unrolling, operation chaining and loop pipelining etc. Furthermore, memory specific optimizations can be applied. For the simplification of accelerator integration, both, shared and streaming memory interfaces are supported. We have also adopted Vivado Design Suite for hardware implementation of interpolation filters.

4.3 HLS Optimizations

For improvement of accelerators performance, HLS tools characterize many optimizations. The basis for these optimizations are the compiler community and some are hardware specific. Current hot area of research for the HLS community is HLS optimizations. In this section we will discuss some of these optimizations in the following paragraphs.

4.3.1 Operation Chaining

This optimization execute operation scheduling for the specified clock period. In a single clock cycle, by the use of this optimization, two combinational operators can be chain together removing the false paths [92].

4.3.2 Bitwidth Optimization

By the use of bit-width optimization, the number of bits needed for the data-path operators are reduced. All the non-functional requirements such as power, area and performance are impacted by the application of this optimization. It does not affect the behaviour of design.

4.3.3 Memory Space Allocation

Distributed block RAMs (BRAMs) are present in FPGAs as form of multiple memory banks. The partitioning and mapping of the software data structures is supported by this structure of the FPGAs. It makes the fast memory accesses implementation at minimum cost. Other way around, the memory ports are very limited in these elements. To configure and customize the memory accesses may need the making of an efficient and optimized architecture based on multi-bank in order to reduce the performance limitation [93].

4.3.4 Loop Optimizations

Loops are the compute-intensive parts of the algorithms. Hardware acceleration for these types of algorithms having compute-intensive loops is significantly important. Loop pipelining is major performance optimization factor for the hardware implementation of the loops. Loop-level parallelism can be exploited by this optimization, if the data dependencies are mitigated, this optimization allows a new loop iteration before finishing of its predecessor. This idea of loop pipelining is related to the software pipelining [94], very long instruction word processors (VLIW) already use this concept. To fully exploit the advantage of parallelism, combination of the loop pipelining and multi-bank architecture is frequently used [93].

4.3.5 Hardware Resource Library

In HLS, meeting the timing requirements for efficient implementation and minimizing the resources usage, it is very necessary to to have the knowledge of how to implement each operation. The given behavioural specification is first inspected by the front-end phase of the HLS implementation. This inspection identifies the characteristics of operations e.g operand type (float and integer), operation type (arithmetic or non-arithmetic), bit-width etc. Some of the operations get benefited from some specific optimizations. For example, division and multiplications by a constant value are transformed into operations of adds and shifts [95], [96] for the improvement of the timing and area. The resulting timing and resources of the circuit are heavily impacted by this methodology. So, for efficient HLS, the composition of this type of library is very crucial.

4.3.6 Speculation and Code Motion

Extraction of parallelism can be done by the use of HLS scheduling techniques. Usually, the parallelism extraction lies within the same control area (same CDFG block), thus resulting in performance limitation of the accelerator specifically in control-intensive systems. A technique known as the Speculation, is used for the code-motion that makes the operations to be shifted with their execution traces. Thus, this code-motion technique anticipate those operations before their conditional constructs [97]-[98].

4.3.7 Exploiting Spatial Parallelism

The spatial parallelism is a hardware acceleration technique. By applying this technique, the hardware may be accelerated as compared to the software implementation. In this technique, for the concurrent execution (spatial parallelism) of the hardware units, multiple hardware units are instantiated.

4.3.8 If-Conversion

A well-known software transformation technique is If-conversion [99]. This technique allows the predicated execution. This means that, the execution of an instruction will only be performed if its predicate evaluates to true. By application of this technique, number of parallel operations are increased. The other advantage is the facilitation of pipelining by the removal of control dependencies in the loop, in turn which may result in the shortening of loop body schedule. On average, 34% improvement is caused by this technique in software [100]. The condition for enabling of the if-conversion is, when branches have balanced requirement of cycles for their execution.

4.4 Xilinx Vivado Design Suite

As stated earlier, AutoPilot [90] initially developed by AutoESL, later Xilinx acquired AutoPilot in 2011 and it becomes Vivado HLS [91]. Xilinx HLS is based on LLVM and released in early 2013. This improved product includes in it, a complete environment for the design with rich characteristics for generation of fine-tune HDL from HLL. Accepting C++, C and SystemC as the input and generating hardware modules in Verilog, VHDL and SystemsC. At the time of compilation it gives the possibility of applying various optimizations such as loop unrolling, operation chaining and loop pipelining etc. Furthermore, memory specific optimizations can be applied. For the simplification of accelerator integration, both, shared and streaming memory interfaces are supported.

The transformation of C code into RTL level implementation in terms of Verilog or VHDL, synthesis of the generated HDL code into Xilinx FPGA, is the flow adopted by the Vivado HLS. Input C code could be in C++, C, SystemC and Open

Computing Language (OpenCL). FPGA supports massively parallel architectures with advantages in cost, performance and power as compared to their counter parts i.e. traditional processors. An overview of Xilinx Vivado high-level synthesis tools flow is presented in this section.

4.4.1 Benefits of High-Level Synthesis

Software and hardware domains can bridged through High-level synthesis. The primary benefits of HLS are listed as follows:

- For hardware designers, improved productivity
- For software designers, improved performance of system
- C-level algorithms development
- C-level functional verification
- C synthesis process control by optimization directives
- Multiple hardware implementations from the same C code by the help of optimization directives
- Portable and readable C code creation

4.4.2 Basics of High-Level Synthesis

Phases of the High-level synthesis are described as follows:

Scheduling

Clock cycle-specific determination of operations occurrence based on:

- Clock frequency or clock cycle length
- Time required to complete the operation, it depends on the target device
- Applied Optimization directives

Binding

Implementation of each operation in corresponding specific hardware is performed by the Binding operation. In HLS, Target device information is used for the optimal implementation of the design.

Control Logic Extraction

For the sequencing of operation in RTL design i.e. to generate finite state machine (FSM), control logic is extracted.

In HLS, C code is synthesized as follows:

- Arguments of top-level function into RTL I/O ports
- C function into RTL level blocks
- By default loops are kept *rolled*
- In final FPGA implementation, arrays into UltraRAM or block RAM

Information about the synthesis performance metrics is contained in synthesis report. These performance metrics described as follows:

- Area: Information about required hardware resources needed for implementation of design e.g. bock RAMS, look-up tables (LUT), DSP48s and registers.
- Latency: Information about the required clock cycles for computation of all values of output.
- Initiation interval (II): Information about the required clock cycles for accepting new inputs.
- Loop iteration latency: Information about the required clock cycles for completion of single iteration of loop.
- Loop initiation interval: Information about the required clock cycles for before next iteration of loop gets start.
- Loop latency: Information about the required clock cycles for all iterations of loop.

4.4.3 Understanding the design flow of Vivado HLS

In Vivado HLS, a C function is synthesized into an IP block. The synthesized IP block can be integrated into a hardware system. Xilinx Vivado HLS is tightly coupled with other Xilinx design tools. It provides broad language support and characteristics for optimal hardware implementation from C algorithm.

The design flow of Vivado HLS is given as follows:

1. C algorithm Compilation, execution (simulation) and debugging.
2. RTL implementation by synthesizing the C algorithm. Here the optimization directives are optionally applied.
3. Synthesis report generation about desgn metrics.
4. RTL verification by a pushbutton flow.
5. RTL implementation packaging into supported IP formats

Inputs and Outputs

The possible inputs of Vivado HLS are listed as follows:

- Function written in C++, C, OpenCL API C kernel or SystemC
- Directives
- Constraints
- C test bench and any associated files

Vivado HLS ouputs are as follows:

- HDL based RTL implementation files The following RTL formats are supported:
  - Verilog (IEEE 1364-2001)
  - VHDL (IEEE 1076-2000)

- Report files

An overview of input and output files of Vivado HLS is shown in Fig. 4.2.

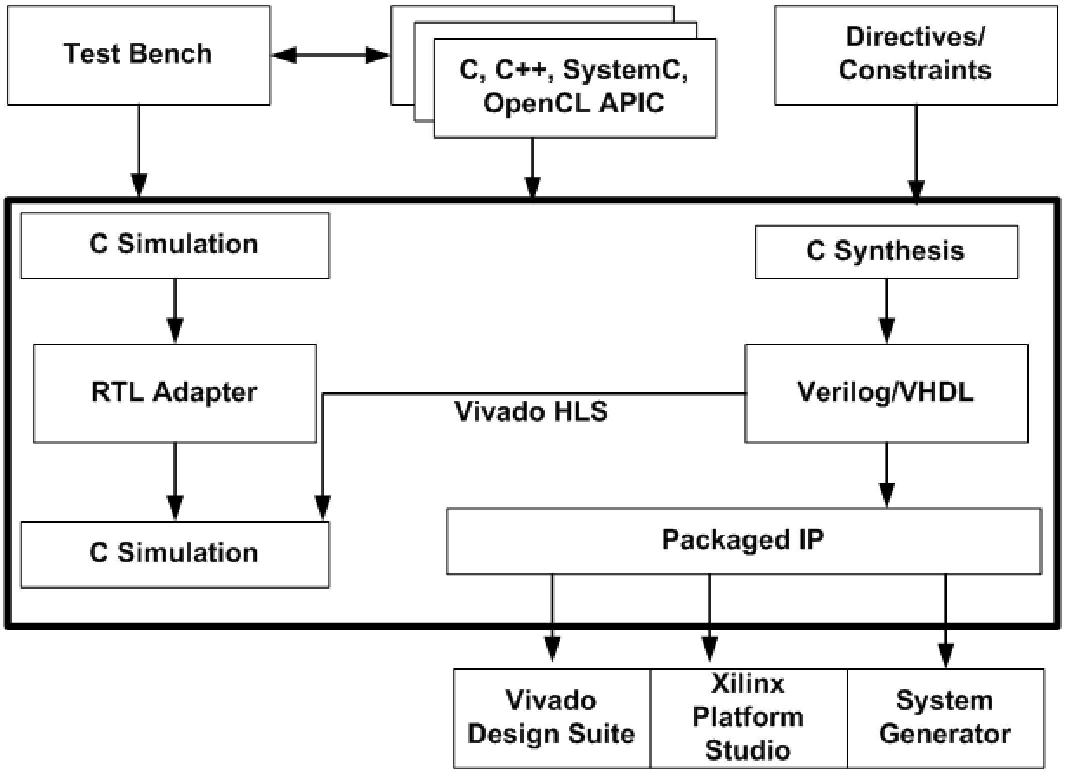


Fig. 4.2 Vivado HLS Design Flow.

Language Support, Test Bench and C Libraries

Top-level function in any C program is called main(). Any sub-function can be specified as top-level function in Vivado HLS. main() cannot be synthesized. Additional rules are as follows:

- Only one top-level function for synthesis is allowed.
- Sub-functions of top-level function are automatically synthesized.
- All the functions to be synthesized must be merged into a single top-level function.

Test Bench

To synthesize functionally correct C function, a test bench is used for functional validation before synthesis, thus improving the productivity,

Language Support

For C simulation/compilation, the following standards are supported:

- C++ (G++ 4.6)
- ANSI-C (GCC 4.6)
- SystemC (IEEE 1666-2006, version 2.2)
- OpenCL API (1.0 embedded profile)

Not supported Language Constructs

The following language constructs are not supported for synthesis:

- Operating system (OS) operations
- Dynamic Memory Allocation

C Libraries

For the FPGA implementation, Vivado HLS contains optimized C libraries. High quality of results (QoR) are achieved by using these libraries. In addition to the standard C language libraries, Vivado HLS provides an extended support for the following C libraries:

- Half-precision (16-bit) floating-point data types
- Arbitrary precision data types
- Video functions
- Math operations

- Maximized usage of shift register LUT (SRL) resources using FPGA resource functions
- Xilinx IP functions, including FFT and FIR

Synthesis, Optimization, and Analysis

A project based on Vivado HLS can holds multiple solutions for a set of C code. Different optimizations and constraints can be applied in each solution. The results based on each solution can be compared in Vivado HLS GUI.

The steps involved in the Vivado HLS design process i.e synthesis, optimization and analysis, are listed as follows:

1. Project creation with an initial solution.
2. Verification of C simulation and execution without error.
3. Design synthesis.
4. Results analysis.

By the analysis of the results, if the design does not meet the requirements, a new solution can be created and synthesized based on new optimization directives and constraints.The process can be repeated until the design performance and requirements are met. The advantage of multiple solutions is moving forward in development and still retaining the old results.

Optimization

Different constraints and optimization directives can be applied to the design in Vivado HLS. Some of them are listed as follows:

- Task Pipelining.
- Latency specification.
- Resources limit specification.
- Override the implied and inherent code dependencies.

- I/O protocol selection.

Analysis

In Vivado HLS, the results can be analysed using the Analysis Perspective. The performance tab in the Analysis Perspective allows to analyse the synthesis results.

RTL Verification

C/RTL co-simulation is supported in Vivado HLS. By using the C/RTL co-simulation infrastructure, RTL verification by simulating the C and RTL design is automatically executed using supported RTL simulator listed as follows:

- ModelSim simulator
- VCS (only supported on Linux operating system)
- Vivado Simulator (XSim) (Vivado Design Suite)
- Riviera (only supported on Linux operating system)
- NCSIm (only supported on Linux operating system)

RTL Export

Final RTL output files can be exported as an IP package in Xilinx Vivado Design Suite. The supported IP formats are listed as follows:

- For use in Vivado Design Suite: Vivado IP Catalog
- For use in Embedded Development Kit (EDK) and for import into Xilinx Platform Studio (XPS): Pcore
- For import directly into the Vivado Design Suite: Synthesized Checkpoint (.dcp)

Chapter 5

HLS Based FPGA Implementation of Interpolation Filters

Video processing systems are becoming more complex thus decreasing the productivity of the hardware designers and the software programmers, producing design productivity gap. To fill this productivity gap, hardware and software fields are bridged through High Level Synthesis (HLS), thus improving the productivity of the hardware designers. One of the most computational intensive parts of High Efficiency Video Coding (HEVC) and H.264/AVC video coding standards is the Interpolation filtering used for sub-pixel interpolation. In this chapter, we present a HLS based FPGA Implementation of sub-pixel Luma and chroma Interpolation of HEVC and sub-pixel Luma interpolation of H.264/AVC, respectively. Xilinx Vivado Design Suite is used for the FPGA implementation of interpolation filtering on Xilinx xc7z020clg481-1 device. The consequent design results in a frame processing speed of 41 QFHD, i.e. 3840x2160@41fps for H.264/AVC sub-pixel Luma interpolation, 46 QFHD for HEVC luma sub-pixel and 48 QFHD for HEVC chroma interpolation. The development time is significantly decreased by the HLS tools.

5.1 Fractional Motion Estimation

Significant storage is needed for uncompressed digital videos. Digital video is handled by high compression and efficient video coding standards, such as HEVC and H.264/AVC. The temporal redundancy present in the video signal is exploited by the process of Motion Compensated Prediction(MCP). MCP reduces the amount of data to be sent to the decoder [101]. We can get rid from large amount of video data by temporal motion prediction. Current block/object location is compared with the previous frame to measure if there exist the same block/object. Hence, reducing the amount of data required to transmit to the video decoder. In MCP, to process the current frame, the similar data/object of the current frame and the previous frame are measured first by the video encoder. For this purpose the frame is divided into blocks of pixels. MCP sends the motion vector as side information to tell the decoder about the similarity between the current frame and the previous frame for prediction. Prediction error is also sent along with the motion vector, for new frame reconstruction. The objects in the consecutive video frames may differ by fractional position i.e. these displacements are continuous. These objects are independent of the sampling grid of the digital video sequence. Fractional motion vector accuracy makes the video encoder efficient and reduce the prediction error [102].

Interpolation filters are used for fractional value motion vector. The design of the interpolation is carried out by keeping in view the important factors such as visual quality, coding efficiency and implementation complexity [103]. H.264/AVC and HEVC video coding standards, support half and quarter pixel accuracy. Interpolation filtering used for sub-pixel interpolation is one of the most computational intensive parts of H.264/AVC and HEVC. Computational complexity of the interpolation filters is about 20% and 25% of total time in 3D-HEVC encoder and decoder, respectively, as reported in our previous work [104]. In industry and academia, HLS is being studied for many years and there exist many operational projects [105]. In this chapter, HLS based FPGA implementation of sub-pixel luma interpolation is presented. Xilinx Vivado HLS tools are used for FPGA implementation of H.264/AVC and HEVC sub-pixel interpolation. HLS has some specific benefits over the conventional RTL based VLSI design. One key benefit is its power to render micro-architectures with specific area vs. performance trade-off for the same behavioural description by surroundings different synthesis choice [106]. A comparison between the HLS based FPGA implementation of sub-pixel interpolation of H.264/AVC and HEVC is also carried out in this chapter.

5.2 H.264/AVC Sub-pixel Interpolation

For 4:2:0 colour format video in H.264/AVC, luma sampling supports the quarter-pel accuracy and chroma sampling support one-eight pixel accuracy of the motion vectors [6]. Motion vector may points to an integer and/or fractional samples position. In the latter case, fractional pixel are generated by interpolation. A one-dimensional 6-tap FIR filter is used for prediction signals at the half-sample value, in vertical and horizontal directions. Average of the sample values at full and half-pixel are used for the quarter sample values generation of the prediction signal.

The luma sub-pixel interpolation process in H.264/AVC is shown in Fig. 5.1. The half pixel values $b_{0,0}$ and $h_{0,0}$ are obtained by applying the 6-tap filter in the horizontal and vertical directions, respectively, as follows:

$$b_{0,0} = (A_{-2,0} - 5 * A_{-1,0} + 20 * A_{0,0} + 20 * A_{1,0} - 5 * A_{2,0} + A_{3,0} + 16) >> 5 \quad (5.1)$$

$$h_{0,0} = (A_{0,-2} - 5 * A_{0,-1} + 20 * A_{0,0} + 20 * A_{0,1} - 5 * A_{0,2} + A_{0,3} + 16) >> 5 \quad (5.2)$$

where $A_{n,0}, A_{0,n}$ with values of $n = -2, -1, 0, 1, 2, 3$, are integer pixels in horizontal and vertical directions, respectively. Intermediate half-pel samples $b'n$ or $h'n$ are used for the calculation of half pixel value $j_{0,0}$, by applying the 6-tap filter in the vertical or horizontal directions, as follows:

$$\begin{aligned} b_n^{'} = b_{n,-2}^{'} - 5 * b_{n,-1}^{'} + 20 * b_{n,0}^{'} + 20 * b_{n,1}^{'} \\ -5 * b_{n,2}^{'} + b_{n,3}^{'} \end{aligned} \quad (5.3)$$

$$j_{0,0} = (b_n^{'} + 512) >> 10 \quad (5.4)$$

where $n = -2, -1, 0, 1, 2, 3$ and $b' = b << 5 - 16$, i.e. we can use the values of b. We can obtain the values of $j_{0,0}$ alternatively, as given by equations 5.5 and 5.6.

$$h_n^{'} = A_{-2,0} - 5 * A_{-1,0} + 20 * A_{0,0} + 20 * A_{1,1} - 5 * A_{2,0} + A_{3,0} \quad (5.5)$$

$$j_{0,0} = (h'_{n,-2} - 5 * h'_{n,-1} + 20 * h'_{n,0} + 20 * h'_{n,1} - 5 * h'_{n,2} + h'_{n,3} + 512) >> 10 \quad (5.6)$$

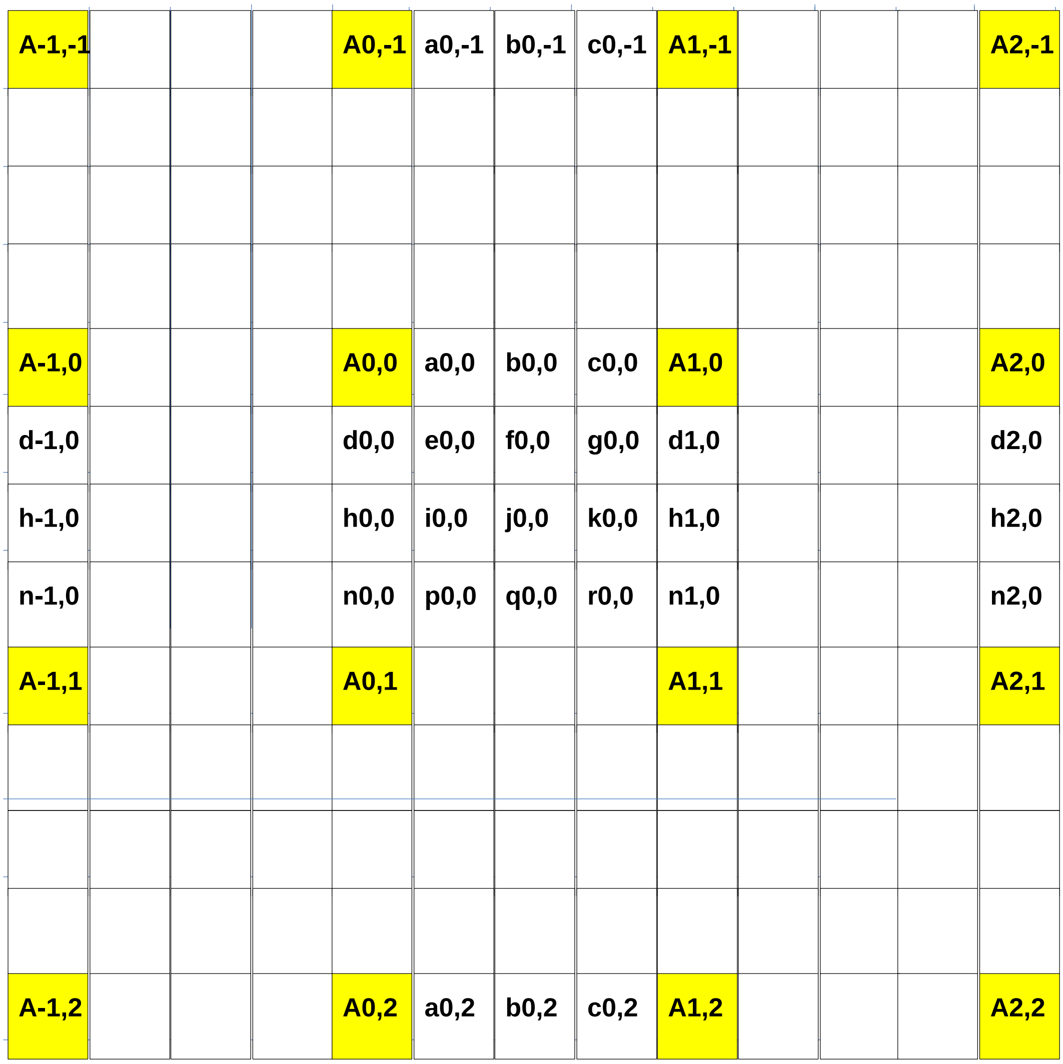
Nearest, half pixel and/or integer pixel averaging is used for the calculation of the quarter-pixel sample. The samples used in the averaging could be both half-pel and a combination of the half-pel and integer-pel samples.

As an example, the following equations shows the method to calculate quarter-pixel samples for some of the quarter-pixel positions i.e. $a_{0,0}, f_{0,0}$ and $e_{0,0}$ out of $a_{0,0}, c_{0,0}, d_{0,0}, n_{0,0}, f_{0,0}, i_{0,0}, k_{0,0}, q_{0,0}, e_{0,0}, g_{0,0}, p_{0,0}$ and $r_{0,0}$:

$$a_{0,0} = (A_{0,0} + b_{0,0} + 1) >> 1 \quad (5.7)$$

$$f_{0,0} = (b_{0,0} + j_{0,0} + 1) >> 1 \quad (5.8)$$

$$e_{0,0} = (b_{0,0} + h_{0,0} + 1) >> 1 \quad (5.9)$$



A-1,-1				A0,-1	a0,-1	b0,-1	c0,-1	A1,-1				A2,-1
A-1,0				A0,0	a0,0	b0,0	c0,0	A1,0				A2,0
d-1,0				d0,0	e0,0	f0,0	g0,0	d1,0				d2,0
h-1,0				h0,0	i0,0	j0,0	k0,0	h1,0				h2,0
n-1,0				n0,0	p0,0	q0,0	r0,0	n1,0				n2,0
A-1,1				A0,1				A1,1				A2,1
A-1,2				A0,2	a0,2	b0,2	c0,2	A1,2				A2,2

Fig. 5.1 Pixel positions for Integer, Luma half and Luma quarter pixels.

5.2.1 HLS based FPGA Implementation

In our proposed design, 13x13 integer pixels are used for the half and quarter pixel interpolation of the 8x8 PU as shown in Fig. 5.2. In Fig. 5.3, the proposed HLS implementation of H.264/AVC luma sub-pixel interpolation is shown. For the larger PU sizes, half and quarter pixel can be interpolated using each 8x8 PU part of the larger block i.e. dividing the larger block in PU sizes of 8x8. 13 integer pixels are given as input to the first half pixel interpolator array $hpi1$ in each clock cycle.

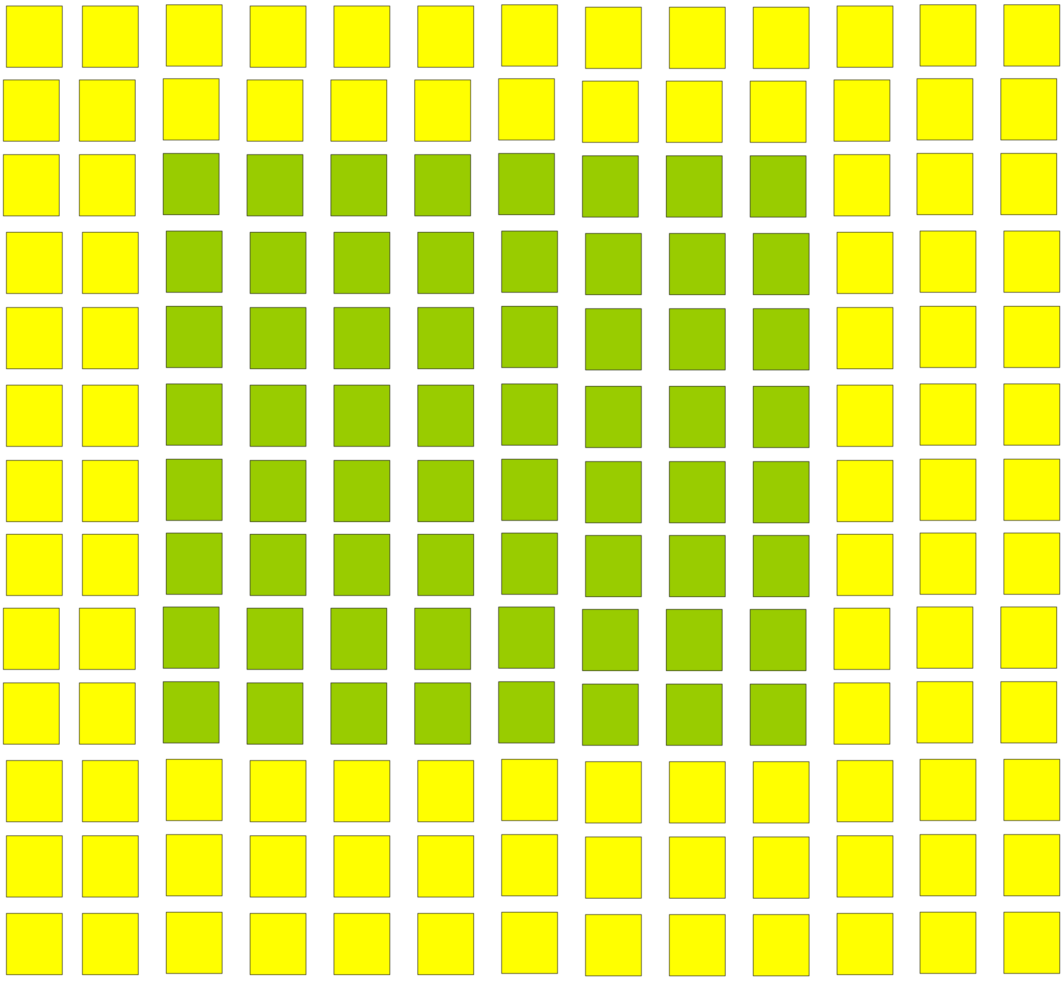


Fig. 5.2 13x13 Pixel Grid for H.264/AVC Luma Interpolation of 8x8 block (where green colour represents the integer pixels block to be interpolated and yellow colour represents the required integer pixels padded to the block to support interpolation).

8 half pixels $b_{0,0}$ are computed in parallel in each clock cycle, so in total it will interpolate 13x8 half pixels in 13 clock cycles. These half pixels are stored into registers for interpolation of the half pixels $j_{0,0}$ or quarter pixels $a_{0,0}$ and $c_{0,0}$. During the interpolation of $b_{0,0}$ half pixels interpolation, 13x13 integer pixels are stored for the half pixel interpolation of the $h_{0,0}$. Then the $h_{0,0}$ half pixels are interpolated using these stored 13x13 integer pixels using $hpi1$, meanwhile, in parallel the $j_{0,0}$ half pixel are interpolated using $hpi2$ from the already available intermediate $b_{0,0}$ half pixels. The half pixels $h_{0,0}$ and $j_{0,0}$ are also stored in the registers for the quarter pixel interpolation. Finally all the $a_{0,0}, c_{0,0}, d_{0,0}, n_{0,0}, f_{0,0}, i_{0,0}, k_{0,0}$, $q_{0,0}, e_{0,0}, g_{0,0}, p_{0,0}$ and

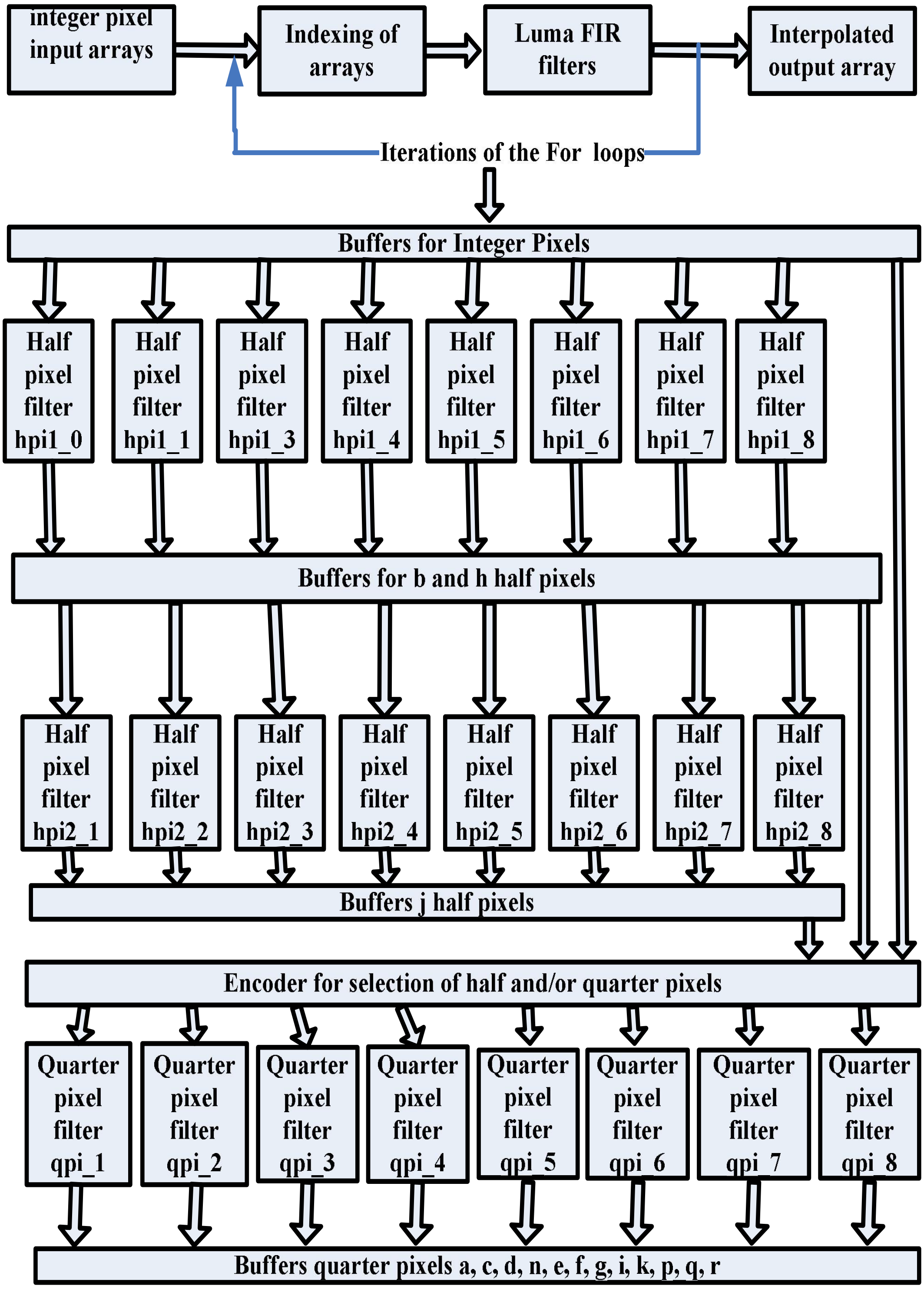

integer pixel input arrays
Indexing of arrays
Luma FIR filters
Interpolated output array
Iterations of the For loops
Buffers for Integer Pixels
Half pixel filter hpi1_0
Half pixel filter hpi1_1
Half pixel filter hpi1_3
Half pixel filter hpi1_4
Half pixel filter hpi1_5
Half pixel filter hpi1_6
Half pixel filter hpi1_7
Half pixel filter hpi1_8
Buffers for b and h half pixels
Half pixel filter hpi2_1
Half pixel filter hpi2_2
Half pixel filter hpi2_3
Half pixel filter hpi2_4
Half pixel filter hpi2_5
Half pixel filter hpi2_6
Half pixel filter hpi2_7
Half pixel filter hpi2_8
Buffers j half pixels
Encoder for selection of half and/or quarter pixels
Quarter pixel filter qpi_1
Quarter pixel filter qpi_2
Quarter pixel filter qpi_3
Quarter pixel filter qpi_4
Quarter pixel filter qpi_5
Quarter pixel filter qpi_6
Quarter pixel filter qpi_7
Quarter pixel filter qpi_8
Buffers quarter pixels a, c, d, n, e, f, g, i, k, p, q, r

Fig. 5.3 HLS implementation of H.264/AVC Luma Sub-pixel.

$r_{0,0}$ quarter pixels are generated using the already computed registered half pixels $b_{0,0}, h_{0,0}, j_{0,0}$ and the 13x13 integer pixels.

Vivado Design Suite is used for HLS based the FPGA implementation of the design. The HLS based design is synthesized to verilog RTL. Vivado HLS tools take $C, C++$ or $SystemC$ codes as input. In our case the C code is applied as input to the vivado HLS tool. The C code is written according to the H.264/AVC reference software video encoder. Vivado HLS provides various optimization techniques called as optimization directives or pragmas. Many variants of the HLS implementation of H.264/AVC luma sub-pixel interpolation are possible depending on the area vs performance trade-off requirements. Design Space Exploration (DSE) of the H.264/AVC luma sub-pixel interpolation is carried out using these optimization directives.

Discussion on Results

Vivado HLS kept the loops as rolled by default. Loops are considered and operated as single sequence of operations defined within the body of the loop. All operations of the loops defined in the body of the loops are synthesized as hardware. So, all iterations of the loops use the same common hardware. Loop UNROLL directive available in the Vivado HLS, unrolls the loops partially or fully, depending on the application requirements. If the application is performance critical, then the loop UNROLL directive can be used to unroll the loops for better optimized hardware in terms of performance by parallel processing, but it will increase the area e.g. if the loops are fully unrolled then the multiple copies of the same hardware will be synthesized. The other directive which we used in our design is PIPELINE. Pipeline directive can be applied to function or loop, it is basically the pipelining. The new inputs can be processed after every N clock cycles. Here N is Initiation Interval (II) i.e. the number of clock cycles after which the new inputs will be processed by the design.

When the pipeline directive is applied, it automatically unrolls all the loops within the scope of the pipeline region i.e. you do not need to apply loop UNROLL directive separately if the pipeline directive is already applied to the scope containing loop. For the parallel processing the data requirement must be satisfied. In our design the arrays are used as input to the HLS tools. Arrays are by default mapped to block RAMs in the Vivado HLS i.e. you can only read or write or both read write at the same time if the block RAM is dual port. So, ARRAY PARTITION directive is used to partition the arrays into individual registers. It makes the data available for the parallel processing.

Two different implementations of the H.264/AVC luma sub-pixel interpolation is carried out using two different techniques for constant multiplication i.e. multiplication using multipliers, multiplication using add and shift operations.

Table 5.1 Resources required for HLS implementation of H.264/AVC Luma Sub-pixel Interpolation using multipliers for multiplication.

Optimization	BRAM18K	DSP48E	FF	LUT	SLICE	Freq. (MHz)	Clock Cycles	Fps
NO OPTIMIZATION	0	0	706	1188	430	128	1489	0.5
LOOP UNROLL	0	0	3011	5084	1670	110	577	1.5
LOOP UNROLL + ARRAY PARTITION	0	0	3451	8653	2655	112	473	2
PIPELINE + ARRAY PARTITION	0	0	10224	27995	8817	102	19	41

Table 5.2 Resources required for HLS implementation of H.264/AVC Luma Sub-pixel Interpolation using add and shift operations for multiplication.

Optimization	BRAM18K	DSP48E	FF	LUT	SLICE	Freq. (MHz)	Clock Cycles	Fps
NO OPTIMIZATION	0	0	302	413	110	129	1432	1
LOOP UNROLL	0	0	2304	3033	422	212	577	3
LOOP UNROLL + ARRAY PARTITION	0	0	2843	4056	748	210	449	3
PIPELINE + ARRAY PARTITION	0	0	11001	12774	2606	102	19	41

Table 5.1 and 5.2 enlist the optimization directives used and the corresponding hardware resources required for HLS implementation using the multipliers as constant multiplication and multiplication by shift and add operations. Mainly three directives are used for the efficient implementation of H.264/AVC Luma interpolation designs. As shown in Table 5.1 and 5.2 , when there is NO OPTIMIZATION directive applied, the latency is much higher i.e. to process 8x8 PU it takes higher clock cycles as compared to the optimized ones. For the optimized design we use the combination of optimization directives such as LOOP UNROLL + ARRAY PARTITION and PIPELINE + ARRAY PARTITION. In both designs the application of optimizations shows significant area vs performance trade-off. In case of constant multiplication using add and shift operations, we have better optimized design in terms of area and performance.

Comparison with Manual RTL implementation

Table 5.3 gives the comparison between HLS and manual RTL implementations of H.264/AVC luma sub-pixel interpolation. It evident that the HLS implementation is more efficient in terms of performance. Even though the other two implementations are VLSI based, we expect the same performance for the FPGA implementations of the corresponding implementations.

Table 5.3 H.264/AVC Luma Sub-pixel HLS vs Manual RTL Implementations.

Comparison Parameter	[107]	[11]	Proposed
Tech.	SMIC 130 nm	130 nm	Xilinx Virtex 7
Slice/Gate Count	75 K	67 K	2606
Freq. (MHz)	340	200	102
Fps	30 QFHD	2160p@30fps	41 QFHD
Design	ME	ME	ME + MC

Comparison with HLS implementation of HEVC

Table 5.4 gives the comparison between HLS implementations of H.264/AVC and HEVC luma sub-pixel interpolation. The proposed implementation takes less area as compared to the HLS implementation of HEVC because the HEVC uses larger interpolation filters and hence larger area. The throughput of the HEVC luma interpolation is also higher because the quarter pixel interpolation is independent of the half pixel interpolation e.g. $a_{0,0}, b_{0,0}, d_{0,0}$ and $h_{0,0}$.

Table 5.4 H.264/AVC vs HEVC Luma Sub-pixel HLS implementation.

Comparison Parameter	Proposed	[108]
Tech.	Xilinx Virtex 7	Xilinx Virtex 6
Slice/Gate Count	2606	4426
Freq. (MHz)	102	168
Fps	41 QFHD	45 QFHD
Design	ME + MC	ME + MC

5.3 HEVC Sub-pixel Interpolation

$A_{i,j}$ upper-case letters within the yellow blocks in Fig. 5.1 represent luma sample positions at full-pixel locations. For the prediction of fractional luma sample values, these integer pixel at ful-pixel locations can be used. White blocks with lower-case letters e.g. $a_{0,0}$, $b_{0,0}$ represent the luma sample positions at quarter-pixel locations.

5.3.1 HEVC Luma Sub-pixel Interpolation

Fractional luma sample positions are computed by Equations (5.10 – 5.24). Fractional luma sample values $a_{0,0}$, $b_{0,0}$, $c_{0,0}$, $d_{0,0}$, $h_{0,0}$ and $n_{0,0}$ are computed by by applying 7 and 8-tap interpolation filters to the integer pixel values specified by Equations (5.10–5.15) as follows:

$$\begin{aligned} a_{0,0} = (&-A_{-3,0} + 4 * A_{-2,0} - 10 * A_{-1,0} + 58 * A_{0,0} + 17 * A_{1,0} \\ &-5 * A_{2,0} + A_{3,0}) >> shift1 \end{aligned} \tag{5.10}$$

$$\begin{aligned} b_{0,0} = (&-A_{-3,0} + 4 * A_{-2,0} - 11 * A_{-1,0} + 40 * A_{0,0} + 40 * A_{1,0} \\ &-11 * A_{2,0} + 4 * A_{3,0} - A_{4,0}) >> shift1 \end{aligned} \tag{5.11}$$

$$\begin{aligned} c_{0,0} = (&A_{-2,0} - 15 * A_{-1,0} + 17 * A_{0,0} + 58 * A_{1,0} - 10 * A_{2,0} \\ &+4 * A_{3,0} - A_{4,0}) >> shift1 \end{aligned} \tag{5.12}$$

$$\begin{aligned} d_{0,0} = (&-A_{0,-3} + 4 * A_{0,-2} - 10 * A_{0,-1} + 58 * A_{0,0} + 17 * A_{0,1} \\ &-5 * A_{0,2} + A_{0,3}) >> shift1 \end{aligned} \tag{5.13}$$

$$\begin{aligned} h_{0,0} = (&-A_{0,-3} + 4 * A_{0,-2} - 11 * A_{0,-1} + 40 * A_{0,0} + 40 * A_{0,1} \\ &-11 * A_{0,2} + 4 * A_{0,3} - A_{0,4}) >> shift1 \end{aligned} \tag{5.14}$$

$$\begin{aligned} n_{0,0} = (&A_{0,-2} - 15 * A_{0,-1} + 17 * A_{0,0} + 58 * A_{0,1} - 10 * A_{0,2} \\ &+4 * A_{0,3} - A_{0,4}) >> shift1 \end{aligned} \tag{5.15}$$

The quarter-pixel values denoted as $e_{0,0}$, $i_{0,0}$, $p_{0,0}$, $f_{0,0}$, $j_{0,0}$, $q_{0,0}$, $g_{0,0}$, $k_{0,0}$ and $r_{0,0}$ are computed by applying 7 and 8-tap filters in vertical direction to the already computed values of $a_{0,i}$, $b_{0,i}$ and $c_{0,i}$ where i = -3 .. 4, as shown by the Equations (5.16 – 5.24)

$$\begin{aligned} e_{0,0} = (&-a_{-3,0} + 4 * a_{-2,0} - 10 * a_{-1,0} + 58 * a_{0,0} + 17 * a_{1,0} \\ &-5 * a_{2,0} + a_{3,0}) >> shift2 \end{aligned} \tag{5.16}$$

$$\begin{aligned} i_{0,0} = (&-a_{-3,0} + 4*a_{-2,0} - 11*a_{-1,0} + 40*a_{0,0} + 40*a_{1,0} \\ &-11*a_{2,0} + 4*a_{3,0} - a_{4,0}) >> shift2 \end{aligned} \tag{5.17}$$

$$\begin{aligned} p_{0,0} = (&a_{-2,0} - 15*a_{-1,0} + 17*a_{0,0} + 58*a_{1,0} - 10*a_{2,0} \\ &+4*a_{3,0} - a_{4,0}) >> shift2 \end{aligned} \tag{5.18}$$

$$\begin{aligned} f_{0,0} = (&-a_{-3,0} + 4*a_{-2,0} - 10*a_{-1,0} + 58*a_{0,0} + 17*a_{1,0} \\ &-5*a_{2,0} + a_{3,0}) >> shift2 \end{aligned} \tag{5.19}$$

$$\begin{aligned} j_{0,0} = (&-a_{-3,0} + 4*a_{-2,0} - 11*a_{-1,0} + 40*a_{0,0} + 40*a_{1,0} \\ &-11*a_{2,0} + 4*a_{3,0} - a_{4,0}) >> shift2 \end{aligned} \tag{5.20}$$

$$\begin{aligned} q_{0,0} = (&a_{-2,0} - 15*a_{-1,0} + 17*a_{0,0} + 58*a_{1,0} - 10*a_{2,0} \\ &+4*a_{3,0} - a_{4,0}) >> shift2 \end{aligned} \tag{5.21}$$

$$\begin{aligned} g_{0,0} = (&-a_{-3,0} + 4*a_{-2,0} - 10*a_{-1,0} + 58*a_{0,0} + 17*a_{1,0} \\ &-5*a_{2,0} + a_{3,0}) >> shift2 \end{aligned} \tag{5.22}$$

$$\begin{aligned} k_{0,0} = (&-a_{-3,0} + 4*a_{-2,0} - 11*a_{-1,0} + 40*a_{0,0} + 40*a_{1,0} \\ &-11*a_{2,0} + 4*a_{3,0} - a_{4,0}) >> shift2 \end{aligned} \tag{5.23}$$

$$\begin{aligned} r_{0,0} = (&a_{-2,0} - 15*a_{-1,0} + 17*a_{0,0} + 58*a_{1,0} - 10*a_{2,0} \\ &+4*a_{3,0} - a_{4,0}) >> shift2 \end{aligned} \tag{5.24}$$

The value of variable shift1 is given by the Equation (5.25) and shift2 = 6.

$$shift1 = BitDepth_Y - 8 \tag{5.25}$$

where $BitDepth_Y$ is the bit depth of luma sample.

5.3.2 HLS based FPGA Implementation of Luma Interpolation

In our proposed design, 15x15 integer pixels are used for the half and quarter pixel interpolation of the 8x8 PU as shown in Fig. 5.4. In Fig. 5.5, the proposed HLS based implementation of HEVC luma sub-pel interpolation is shown. For the larger PU sizes, half and quarter pixel can be interpolated using each 8x8 PU part of the larger block i.e. dividing the larger block in PU sizes of 8x8. 15 integer pixels are given as input to the array of sub-pixel interpolator filter i.e. $FilterSetabc1$–$FilterSetabc8$ in

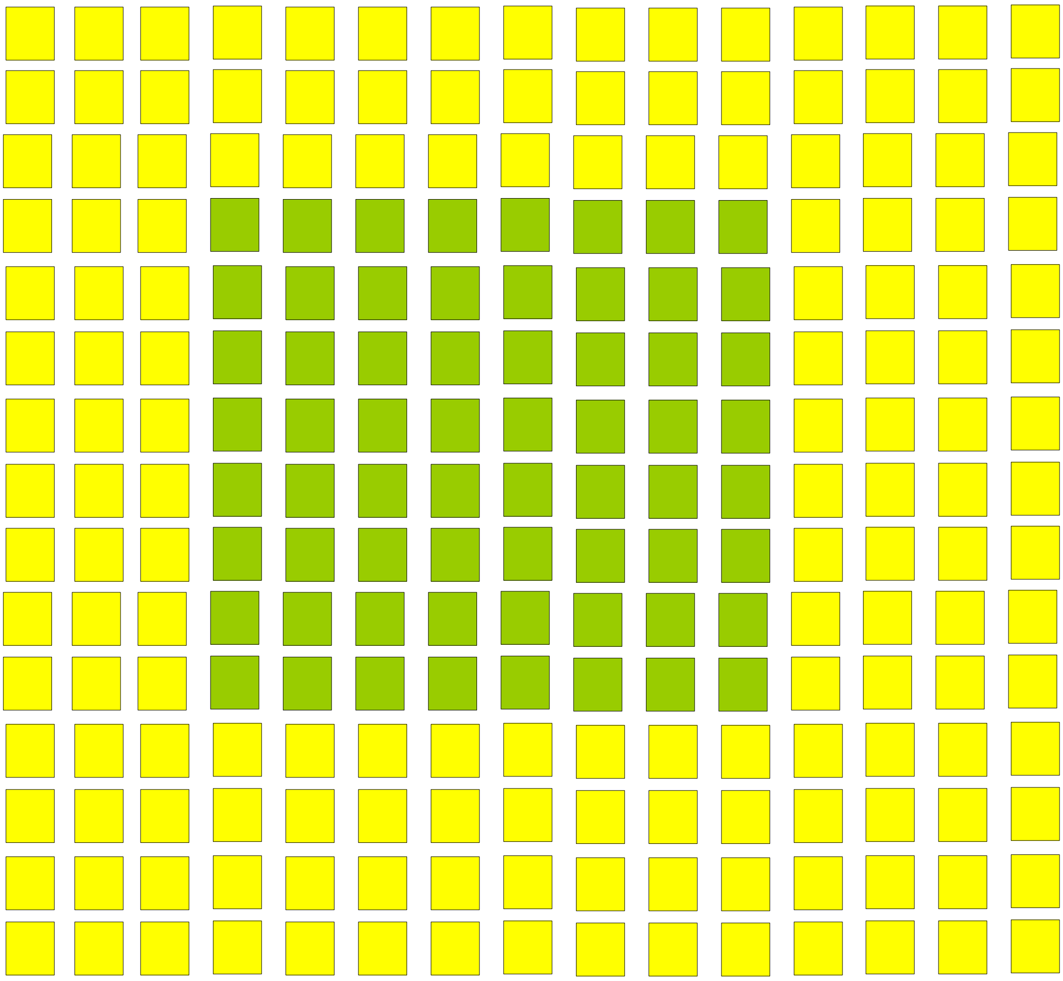


Fig. 5.4 15x15 Pixel Grid for HEVC Luma Interpolation of 8x8 block (where green colour represents the integer pixels block to be interpolated and yellow colour represents the required integer pixels padded to the block to support interpolation).

each clock cycle. 24 sub-pixels i.e. $8a, 8b, 8c$ are computed in parallel in each clock cycle, so in total it will interpolate 15x24 half pixels in 15 clock cycles. These half pixels are stored into registers for computing the half pixels e.g. $e_{0,0}$, $f_{0,0}$, $j_{0,0}$ etc. 15x15 integer pixels are stored for the half pixel interpolation of the $a_{0,0}, b_{0,0}, c_{0,0}$ etc. Then the $d_{0,0}, h_{0,0}, n_{0,0}$ half pixels are interpolated using these stored 15x15 integer pixels using the same filter set. Finally the half pixels e.g. $e_{0,0}, f_{0,0}, j_{0,0}$ etc. are interpolated using the already stored half pixels $a_{0,0}, b_{0,0}, c_{0,0}$.

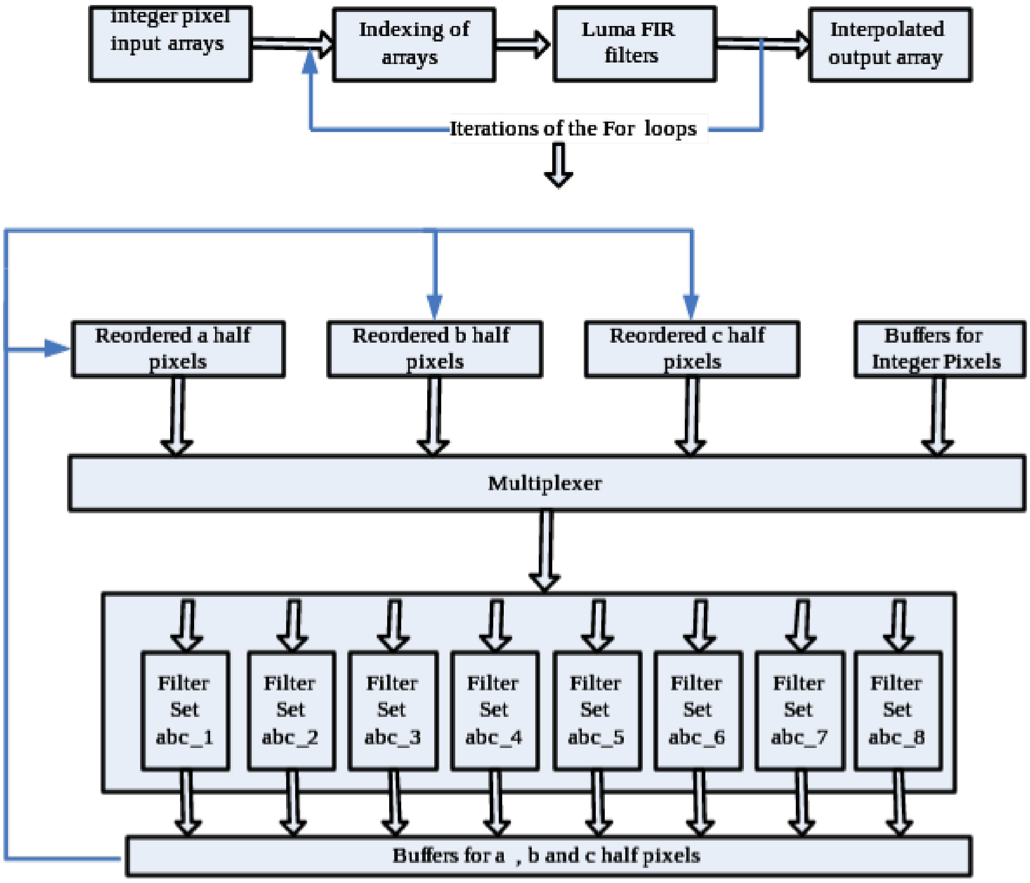


Fig. 5.5 HLS implementation of HEVC Luma Sub-pixel.

Two different implementations of the HEVC luma sub-pixel interpolation is carried out using two different techniques for constant multiplication i.e. multiplication using multipliers, multiplication using add and shift operations.

Table 5.5 and 5.6 enlist the optimization directives used and the corresponding hardware resources required for HLS implementation using the multipliers as constant multiplication and multiplication by shift and add operations. Mainly three directives are used for the efficient implementation of HEVC Luma interpolation designs. As shown in Table 5.5 and 5.6 , when there is NO OPTIMIZATION directive applied, the latency is much higher i.e. to process 8x8 PU it takes higher clock cycles as compared to the optimized ones. For the optimized design we use the combination of optimization directives such as LOOP UNROLL + ARRAY PARTITION and PIPELINE + ARRAY PARTITION. In both designs the application of optimizations shows significant area vs performance trade-off. In case of constant multiplication using add and shift operations, we have better optimized design in terms of area and performance.

Table 5.5 Resources required for HLS based HEVC luma implementation using multipliers for multiplication.

Optimization	BRAM18K	DSP48E	FF	LUT	SLICE	Freq. (MHz)	Clock Cycles	Fps
NO OPTIMIZATION	0	0	1221	1845	718	218	1505	1
LOOP UNROLL	0	0	4132	14031	2167	150	190	6
LOOP UNROLL + ARRAY PARTITION	0	0	8201	17215	3356	165	130	10
PIPELINE + ARRAY PARTITION	0	0	11490	29534	9315	165	59	21

Table 5.6 Resources required for HLS based HEVC luma implementation using add and shift operations for multiplication.

Optimization	BRAM18K	DSP48E	FF	LUT	SLICE	Freq. (MHz)	Clock Cycles	Fps
NO OPTIMIZATION	0	0	719	1243	325	210	966	2
LOOP UNROLL	0	0	3203	8598	1388	180	190	7
LOOP UNROLL + ARRAY PARTITION	0	0	6456	12766	1890	165	88	14
PIPELINE + ARRAY PARTITION	0	0	11122	14452	3477	165	28	46

Comparison with Manual RTL implementation

Table 5.7 gives the comparison between HLS and manual RTL implementations of HEVC luma sub-pixel interpolation. It is evident that the HLS implementation is more efficient in terms of performance. Even though the other three implementations are VLSI based, we expect the same performance for the FPGA implementations of the corresponding implementations.

Table 5.7 HEVC luma sub-pixel HLS vs manual RTL Implementations.

Comparison Parameter	[10]	[109]	[110]	[111]	Proposed
Technology	Xilinx Virtex 6	90nm	150nm	90nm	Xilinx Virtex 7
Slice/Gate Count	1597	32.5 K	30.5 K	224 K	3477
Frq. (MHz)	200	171	312	333	165
Fps	30 QFHD	60 QFHD	30 QFHD	30 QFHD	46 QFHD
Design	ME + MC	MC	ME + MC	ME + MC	ME + MC

5.3.3 HEVC Chroma Sub-pixel Interpolation

$B_{i,j}$ upper-case letters within the shaded blocks in Fig. 5.6 represent chroma sample positions at full-pixel locations. For the prediction of fractional chroma sample values, these integer pixel at full-pixel locations can be used. Un-shaded blocks with lower-case letters e.g. $ab_{0,0}$, $ac_{0,0}$ represent the chroma sample positions at eight-pixel locations.

Fractional chroma sample positions are computed by Equations (5.26 – 5.32). Fractional chroma sample values $ab_{0,0}$, $ac_{0,0}$, $ad_{0,0}$, $ae_{0,0}$, $af_{0,0}$, $ag_{0,0}$ and $ah_{0,0}$ are computed by by applying 4-tap interpolation filters to the integer pixel values specified by Equations (5.10–5.15) as follows:

$$ab_{0,0} = (-2*B_{-1,0} + 58*B_{0,0} + 10*B_{1,0} - 2*B_{2,0}) >> shift1 \tag{5.26}$$

$$ac_{0,0} = (-4*B_{-1,0} + 54*B_{0,0} + 16*B_{1,0} - 2*B_{2,0}) >> shift1 \tag{5.27}$$

$$ad_{0,0} = (-6*B_{-1,0} + 46*B_{0,0} + 28*B_{1,0} - 4*B_{2,0}) >> shift1 \tag{5.28}$$

$$ae_{0,0} = (-4*B_{-1,0} + 36*B_{0,0} + 36*B_{1,0} - 4*B_{2,0}) >> shift1 \tag{5.29}$$

$$af_{0,0} = (-4*B_{-1,0} + 28*B_{0,0} + 46*B_{1,0} - 6*B_{2,0}) >> shift1 \tag{5.30}$$

$$ag_{0,0} = (-2*B_{-1,0} + 16*B_{0,0} + 54*B_{1,0} - 4*B_{2,0}) >> shift1 \tag{5.31}$$

$$ah_{0,0} = (-2*B_{-1,0} + 10*B_{0,0} + 58*B_{1,0} - 2*B_{2,0}) >> shift1 \tag{5.32}$$

Fractional chroma sample values $ba_{0,0}$, $ca_{0,0}$, $da_{0,0}$, $ea_{0,0}$, $fa_{0,0}$, $ga_{0,0}$ and $ha_{0,0}$ are computed by by applying 4-tap interpolation filters to the integer pixel values specified by Equations (5.33–5.39) as follows:

$$ba_{0,0} = (-2*B_{0,-1} + 58*B_{0,0} + 10*B_{0,1} - 2*B_{0,2}) >> shift1 \tag{5.33}$$

$$ca_{0,0} = (-4*B_{0,-1} + 54*B_{0,0} + 16*B_{0,1} - 2*B_{0,2}) >> shift1 \tag{5.34}$$

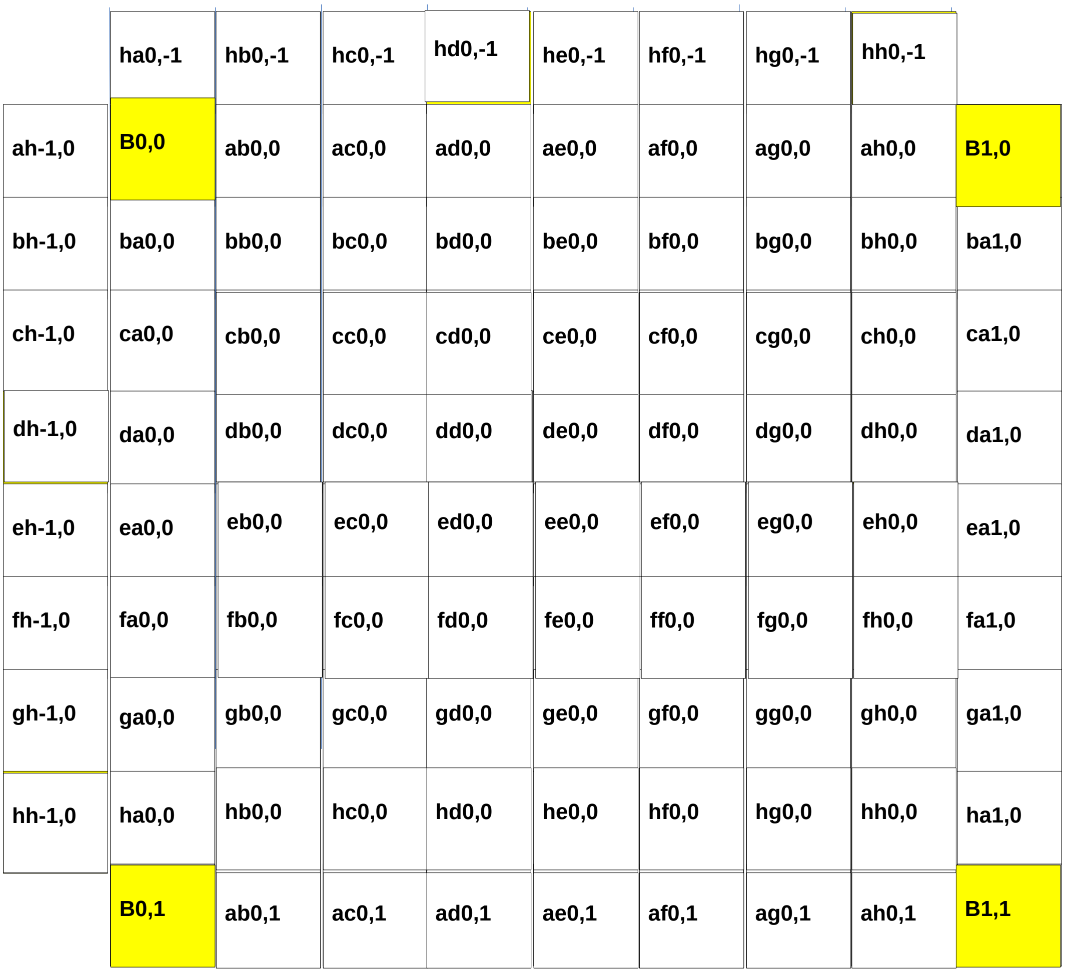
$$da_{0,0} = (-6*B_{0,-1} + 46*B_{0,0} + 28*B_{0,1} - 4*B_{0,2}) >> shift1 \tag{5.35}$$

$$ea_{0,0} = (-4*B_{0,-1} + 36*B_{0,0} + 36*B_{0,1} - 4*B_{0,2}) >> shift1 \tag{5.36}$$

$$fa_{0,0} = (-4*B_{0,-1} + 28*B_{0,0} + 46*B_{0,1} - 6*B_{0,2}) >> shift1 \tag{5.37}$$

$$ga_{0,0} = (-2*B_{0,-1} + 16*B_{0,0} + 54*B_{0,1} - 4*B_{0,2}) >> shift1 \tag{5.38}$$

$$ha_{0,0} = (-2*B_{0,-1} + 10*B_{0,0} + 58*B_{0,1} - 2*B_{0,2}) >> shift1 \tag{5.39}$$



	ha0,-1	hb0,-1	hc0,-1	hd0,-1	he0,-1	hf0,-1	hg0,-1	hh0,-1	
ah-1,0	B0,0	ab0,0	ac0,0	ad0,0	ae0,0	af0,0	ag0,0	ah0,0	B1,0
bh-1,0	ba0,0	bb0,0	bc0,0	bd0,0	be0,0	bf0,0	bg0,0	bh0,0	ba1,0
ch-1,0	ca0,0	cb0,0	cc0,0	cd0,0	ce0,0	cf0,0	cg0,0	ch0,0	ca1,0
dh-1,0	da0,0	db0,0	dc0,0	dd0,0	de0,0	df0,0	dg0,0	dh0,0	da1,0
eh-1,0	ea0,0	eb0,0	ec0,0	ed0,0	ee0,0	ef0,0	eg0,0	eh0,0	ea1,0
fh-1,0	fa0,0	fb0,0	fc0,0	fd0,0	fe0,0	ff0,0	fg0,0	fh0,0	fa1,0
gh-1,0	ga0,0	gb0,0	gc0,0	gd0,0	ge0,0	gf0,0	gg0,0	gh0,0	ga1,0
hh-1,0	ha0,0	hb0,0	hc0,0	hd0,0	he0,0	hf0,0	hg0,0	hh0,0	ha1,0
	B0,1	ab0,1	ac0,1	ad0,1	ae0,1	af0,1	ag0,1	ah0,1	B1,1

Fig. 5.6 Chroma sample grid for eight sample interpolation

Fractional chroma sample values $bV_{0,0}$, $cV_{0,0}$, $dV_{0,0}$, $eV_{0,0}$, $fV_{0,0}$, $gV_{0,0}$ and $hV_{0,0}$ for V being replaced by b, c, d, e, f, g and h, respectively, are computed by applying 4-tap interpolation filters to the intermediate values $aV_{0,i}$ with i = -1..2 in the vertical direction as given by Equations (5.40–5.46) as follows:

$$bV_{0,0} = (-2*aV_{0,-1} + 58*aV_{0,0} + 10*aV_{0,1} - 2*aV_{0,2}) >> shift2 \tag{5.40}$$

$$cV_{0,0} = (-4*aV_{0,-1} + 54*aV_{0,0} + 16*aV_{0,1} - 2*aV_{0,2}) >> shift2 \tag{5.41}$$

$$dV_{0,0} = (-6*aV_{0,-1} + 46*aV_{0,0} + 28*aV_{0,1} - 4*aV_{0,2}) >> shift2 \tag{5.42}$$

$$eV_{0,0} = (-4 * aV_{0,-1} + 36 * aV_{0,0} + 36 * aV_{0,1} - 4 * aV_{0,2}) >> shift2 \tag{5.43}$$

$$fV_{0,0} = (-4 * aV_{0,-1} + 28 * aV_{0,0} + 46 * aV_{0,1} - 6 * aV_{0,2}) >> shift2 \tag{5.44}$$

$$gV_{0,0} = (-2 * aV_{0,-1} + 16 * aV_{0,0} + 54 * aV_{0,1} - 4 * aV_{0,2}) >> shift2 \tag{5.45}$$

$$hV_{0,0} = (-2 * aV_{0,-1} + 10 * aV_{0,0} + 58 * aV_{0,1} - 2 * aV_{0,2}) >> shift2 \tag{5.46}$$

The value of variable shift1 is given by the Equation (5.47) and shift2 = 6.

$$shift1 = BitDepth_C - 8 \tag{5.47}$$

where $BitDepth_C$ is the bit depth of chroma sample.

5.3.4 HLS based FPGA Implementation of Chroma Interpolation

In our proposed design, 7x7 integer pixels are used for the eight-pixel interpolation of the 4x4 chroma PU as shown in Fig. 5.7. In Fig. 5.8, the proposed HLS based implementation of HEVC chroma sub-pel interpolation is shown. For the larger PU sizes, eight-pixel can be interpolated using each 4x4 PU part of the larger block i.e. dividing the larger block in PU sizes of 4x4. 7 integer pixels are given as input to the array of sub-pixel interpolator filter i.e. $FilterSetbcdefgh_1$-$FilterSetbcdefgh_4$ in each clock cycle. 28 sub-pixels i.e. $4a, 4b, 4c, 4d, 4f, 4g$ and $4h$ are computed in parallel in each clock cycle, so in total it will interpolate 7x28 sub pixels in 7 clock cycles. These sub-pixels are stored into registers for computing the half pixels e.g.$bb_{0,0}, bc_{0,0}, bh_{0,0}$ etc. 7x7 integer pixels are stored for the sub-pixel interpolation of the $bb_{0,0}, bc_{0,0}, bh_{0,0}$ etc. Then the $ba_{0,0}$–$ha_{0,0}$ sub-pixels are interpolated using these stored 7x7 integer pixels using the same filter set. Finally the sub-pixels e.g. $e_{0,0}, f_{0,0}, j_{0,0}$ etc. are interpolated using the already stored half pixels $ab_{0,0} - ah_{0,0}$.

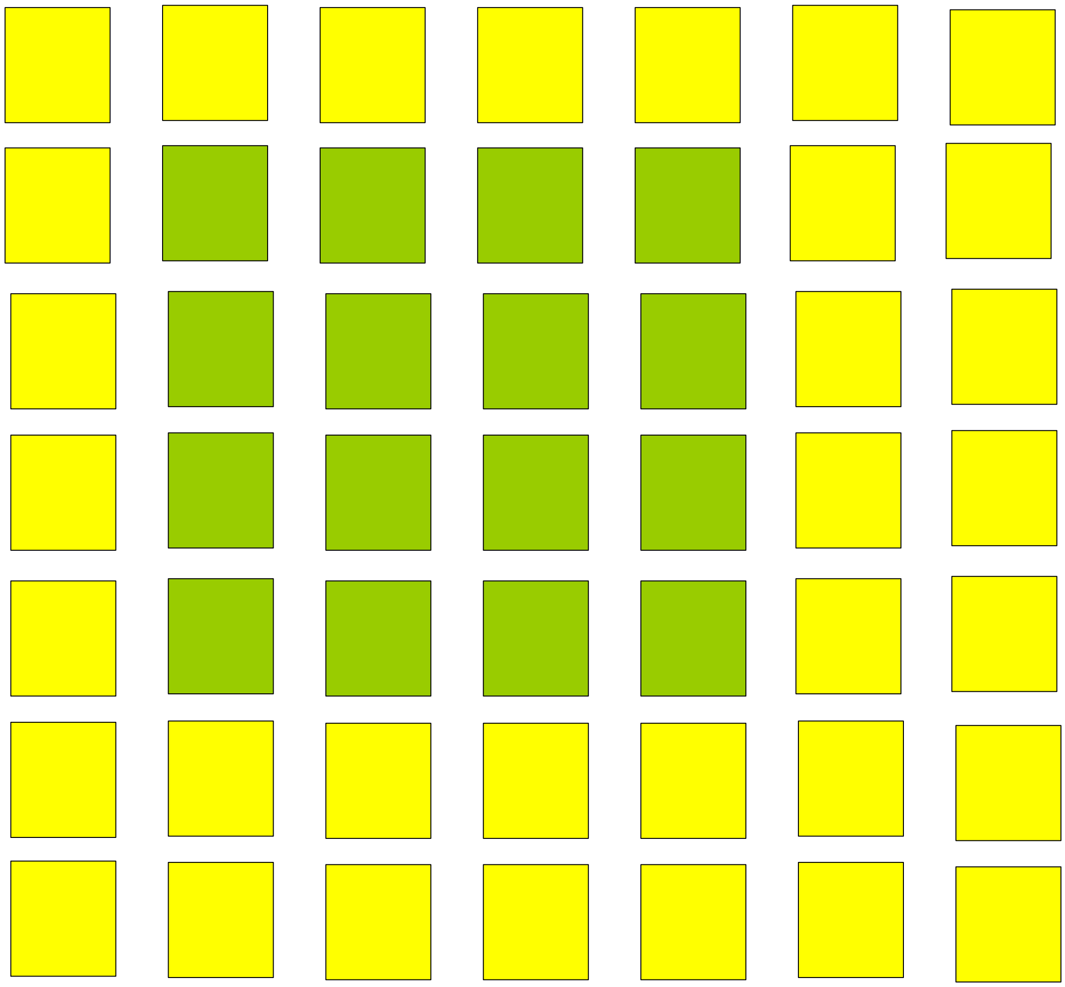


Fig. 5.7 7x7 Pixel Grid for HEVC chroma Interpolation of 4x4 block (where green colour represents the integer pixels block to be interpolated and yellow colour represents the required integer pixels padded to the block to support interpolation).

Two different implementations of the HEVC chroma sub-pixel interpolation is carried out using two different techniques for constant multiplication i.e. multiplication using multipliers, multiplication using add and shift operations.

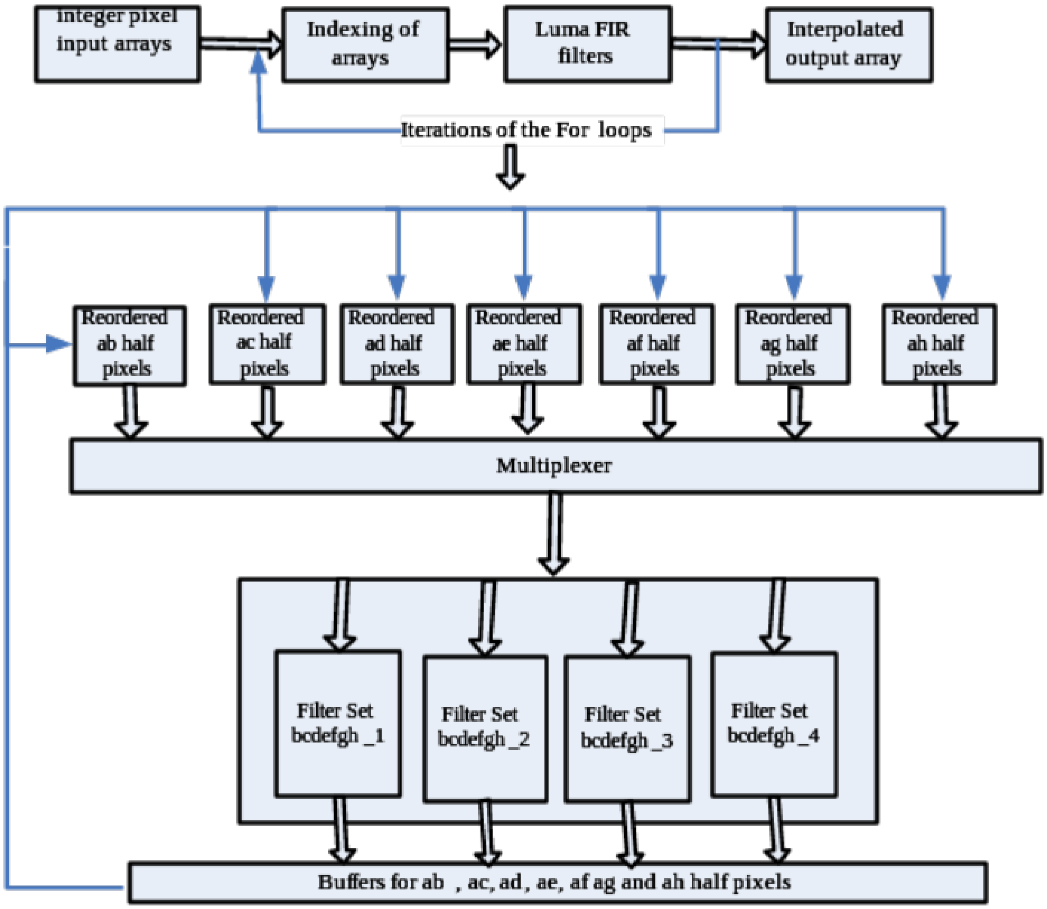


Fig. 5.8 HLS implementation of HEVC Chroma Sub-pixel.

Table 5.8 Resources required for HLS based HEVC chroma implementation using multipliers for multiplication.

Optimization	BRAM18K	DSP48E	FF	LUT	SLICE	Freq. (MHz)	Clock Cycles	Fps
NO OPTIMIZATION	0	0	615	1020	313	220	1424	2
LOOP UNROLL	0	0	2144	7010	1077	162	178	7
LOOP UNROLL + ARRAY PARTITION	0	0	4671	9156	1796	169	116	11
PIPELINE + ARRAY PARTITION	0	0	6723	15884	5358	169	53	24

Table 5.9 Resources required for HLS based HEVC chroma implementation using add and shift operations for multiplication.

Optimization	BRAM18K	DSP48E	FF	LUT	SLICE	Freq. (MHz)	Clock Cycles	Fps
NO OPTIMIZATION	0	0	321	654	325	215	622	3
LOOP UNROLL	0	0	1603	8598	745	176	145	9
LOOP UNROLL + ARRAY PARTITION	0	0	3288	12766	967	169	79	17
PIPELINE + ARRAY PARTITION	0	0	5986	14452	1752	169	27	48

Table 5.8 and 5.9 enlist the optimization directives used and the corresponding hardware resources required for HLS implementation using the multipliers as constant multiplication and multiplication by shift and add operations. Mainly three directives are used for the efficient implementation of HEVC Chroma interpolation designs. As shown in Table 5.8 and 5.9 , when there is NO OPTIMIZATION directive applied, the latency is much higher i.e. to process 4x4 PU it takes higher clock cycles as compared to the optimized ones. For the optimized design we use the combination of optimization directives such as LOOP UNROLL + ARRAY PARTITION and PIPELINE + ARRAY PARTITION. In both designs the application of optimizations shows significant area vs performance trade-off. In case of constant multiplication using add and shift operations, we have better optimized design in terms of area and performance.

Comparison with Manual RTL implementation

Table 5.10 gives the comparison between HLS and manual RTL implementations of HEVC chroma sub-pixel interpolation. It is evident that the HLS implementation is more efficient in terms of performance. Even though the other implementation is VLSI based, we expect the same performance for the FPGA implementations of the corresponding implementations.

Table 5.10 HEVC Chroma sub-pixel HLS vs manual RTL Implementations.

Comparison Parameter	[110]	Proposed
Technology	150nm	Xilinx Virtex 7
Slice/Gate Count	1132	1752
Frq. (MHz)	312	169
Fps	30 QFHD	48 QFHD
Design	ME + MC	ME + MC

5.3.5 Summary: HLS vs manual RTL Implementations

In this work, hardware implementation of the HEVC interpolation filters and H.264/AVC luma interpolation based on HLS design flow is presented. The throughput of HLS accelerator is 41 QFHD, i.e. 3840x2160@41fps for H.264/AVC sub-pixel Luma interpolation, 46 QFHD for HEVC luma sub-pixel and 48 QFHD for HEVC chroma interpolation. It achieves almost the same performance as the manual implementation with half of development time. The comparison of the Xilinx Vivado HLS based implementation with already available RTL-style hardware implementations built using Xilinx System Generator is presented. Both hardware designs were implemented using Xilinx Vivado Suite as stand-alone cores by using Xilinx Virtex 7 xc7z020clg481-1 device. By optimizing and refactoring the C algorithmic model, we became able to implement a design that has almost the same performance as the reference implementation. The design time required for HLS based implementation is significant low about half of the design time required by the manual RTL implementation.

Our design is semi-automated by using Vivado HLS tools. Other designs were developed using manual RTL Verilog/VHDL typical design flow. Design time for Xilinx Vivado HLS was extracted from control log of source code. This time shows the design time taken needed by a designer who has expertise in tool not a domain expert. This means, the designer takes an unfamiliar code, use the tool to implement it as first design, refactor the code to have an optimized desired architecture. Discover the RTL improvements made in the algorithm by reverse engineering the original RTL code and perform design-space exploration. Originally, the algorithm itself do not have those improvements in software model. The throughput in terms of fps of the manual design almost the same as that of the HLS design.

5.3.6 Design Time Reduction

For comparison purpose we are designing the same interpolation filters using both design methodologies. The breakdown of each methodology for the purpose of evaluation and comparison of development times is shown in Fig. 5.9. For the same hardware functionality both the design flows have different design steps with different development time. Vivado HLS based design allows implementing the architecture in about 3 months. The manual design will take about 6 months to implement the final hardware design. A trade-off between performance and design time is observed in both methodologies: It is observed that the manual design flow takes almost double the design time than HLS design flow.

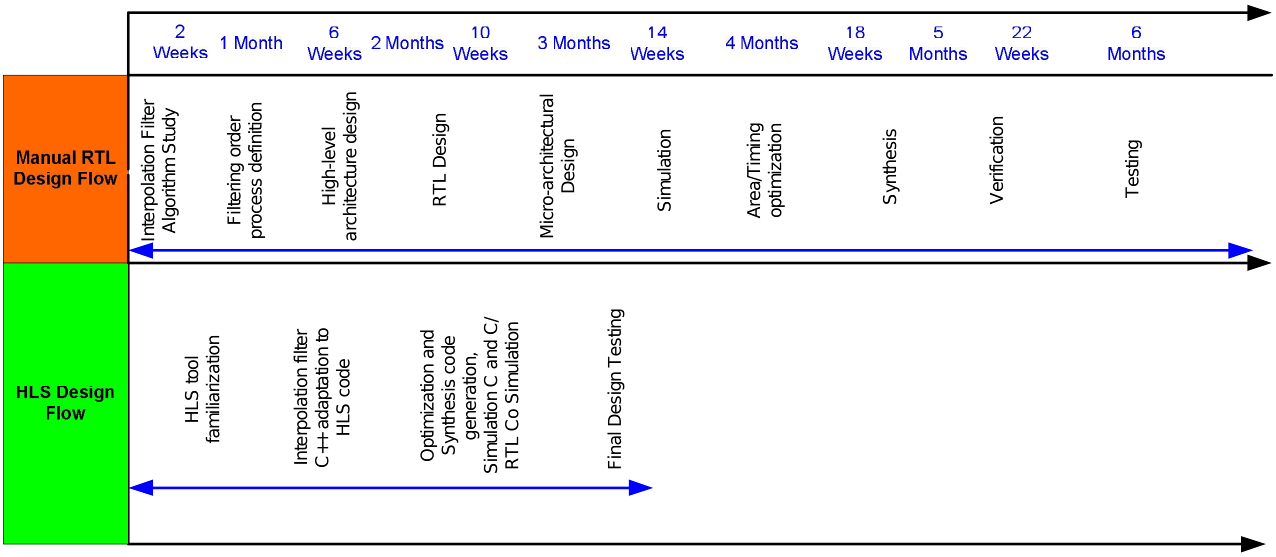

2 Weeks
1 Month
6 Weeks
2 Months
10 Weeks
3 Months
14 Weeks
4 Months
18 Weeks
5 Months
22 Weeks
6 Months
Manual RTL Design Flow
Interpolation Filter Algorithm Study
Filtering order process definition
High-level architecture design
RTL Design
Micro-architectural Design
Simulation
Area/Timing optimization
Synthesis
Verification
Testing
HLS Design Flow
HLS tool familiarization
Interpolation filter C++adaptation to HLS code
Optimization and Synthesis code generation, Simulation C and C/RTL Co Simulation
Final Design Testing

Fig. 5.9 Design time comparison HLS vs Manual RTL Design.

Thus we were able to draw the conclusion that High Level Synthesis (HLS) tools provides shorter development time by automatically generating the hardware implementations still working at a higher level of software abstraction [112]. Furthermore, HLS design flow has large design-space exploration of different performance and area trade-off. HLS flow is also efficient in terms of simulation, documentation design, coding, debugging and reuse. Instead, manual design needs long time to generate a single hardware architecture that requires further redesign effort if time or area constraints are not fulfilled.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Time-to-market is the critical factor for digital systems, thus increasing the requirement of FPGA platforms. FPGA platforms help to avoid manufacturing cycles and chip design time. Design time reduction at the cost of increased power, performance or cost is acceptable for the designers. Latest HLS tools provide option to designers to consider HLS tools as hardware implementation due to significant design time reduction and comparable Quality-of-Results (QoR) as manual RTL design.

6.1.1 Hardware Implementation: HLS vs Manual RTL

Factors on which FPGA system design's time-to-market depend, includes development boards, reference designs and FPGA devices themselves. The primary aim of the HLS design is to increase the designer productivity by implementing the architecture for new algorithms with more ease. Reduction of time-to-market depends on many factors including design time and functional form of the design in terms of integration into a working system. System integration, embedded software and the verification are all concerned parts of the system-level design integration [113].

A trade-off between design time and performance is presented and discussed with respect to both methodologies: The HLS design flow time is less than the half of manual design flow time.

For verification and testing steps, manual design needs more time and effort since possibility of errors is considerable in this kind of design, unlike automated design flows that minimize errors prone. Generation of RTL code is automatized in proposed HLS design. But the generation of the interface between processor and accelerator is still manual configured.

Verification productivity is the key factor in the adoption of high-level synthesis methodologies for hardware implementation. Design simulation at high-level i.e. C level is much faster than the simulation at RTL level implementation. This does not mean that the RTL verification is no more needed, instead it states that the design time can be significantly reduced by reducing the verification-debug cycles. In RTL, the verification-debug cycles take a lot of design time. The HLS users can achieve almost two times improvement in verification process with almost the same design performance.

6.2 Future Work

Synthesized View Distortion Change (SVDC) using Renderer model (TRenSingleModelC Class), was identified as the second most computational intensive part of the 3D-HEVC encoder. To the best of our information, still there is no hardware implementation available in the literature for SVDC. As a future work, for the hardware implementation of this critical part, we have analysed the high-level hardware architecture for the renderer model as given in the following paragraphs. This analysis gives the high-level detail about the possible hardware blocks of the renderer model. In next step, we plan to have manual-RTL based hardware implementation vs HLS implementation of the renderer model.

6.2.1 3D-HEVC Renderer Model

Depth maps are used for virtual view synthesis. Distortion in depth maps coding affect the quality of synthesized views. SVDC due to the coding of current depth block is given by

$$\begin{aligned}SVDC &= D_{distCB} - D_{orgCB} \\ &= \textstyle\sum_{(x,y)\in S}\left[S_{T,distCB}(x,y) - S_{T,Reforg}(x,y)\right]^2 \\ &- \textstyle\sum_{(x,y)\in S}\left[S_{T,orgCB}(x,y) - S_{T,Reforg}(x,y)\right]^2\end{aligned} \tag{6.1}$$

where $S_{T,Reforg}$ is the reference view synthesized from original input left, right textures $S_{T,l}$, $S_{T,r}$ and depth maps $S_{D,l}$, $S_{D,r}$. $S_{T,orgCB}$ and $S_{T,distCB}$ are views synthesized by using original S_{orgCB} (indicated in yellow) and distorted S_{distCB} (indicated in red) data of current depth block as shown in the Fig. 6.1. T, D, l, r, x , y and S shows texture, depth, left, right, horizontal component, vertical component of pixel and the set of pixels in the synthesized view, respectively. D_{distCB}, D_{orgCB} are distortion while using distorted and original current depth block, respectively. $SVDC$ is the difference between D_{distCB}, and D_{orgCB}. As shown in the Fig. 6.1, as an example, if the current depth frame is divided into four depth blocks $B0$, $B1$, $B2$ and $B3$, where the $B0$ and $B1$ are already coded blocks, $B3$ is the current block to be coded and $B4$ is the original block to be coded after coding of $B3$ is finished. SSD unit computes the Sum of Squared Differences of $S_{T,Reforg}$ and $S_{T,distCB}$, $S_{T,orgCB}$.

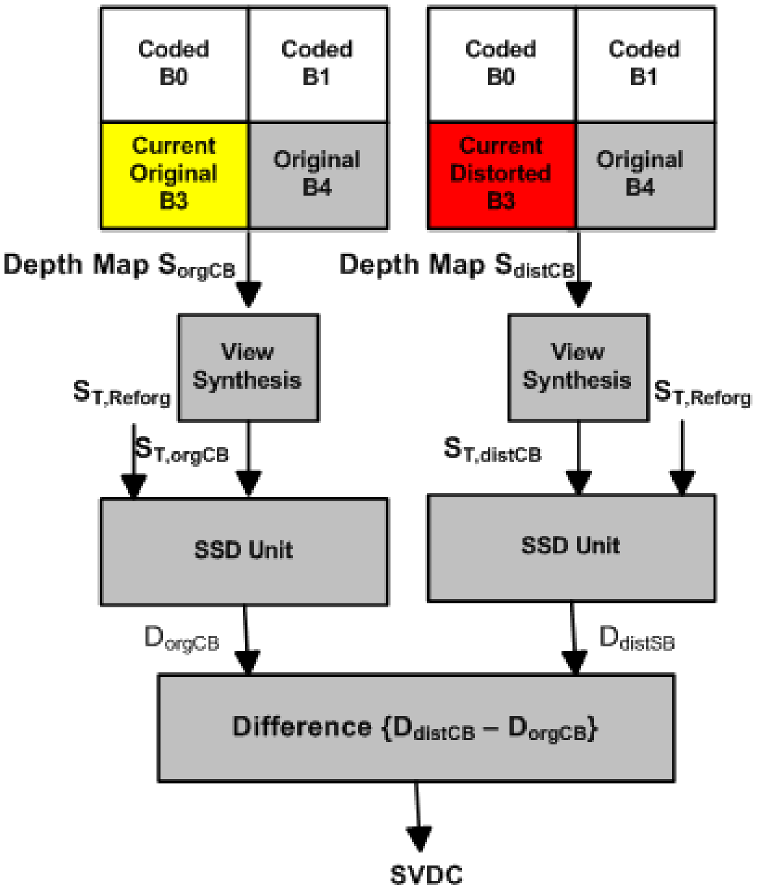


Fig. 6.1 Block Diagram of SVDC.

For the efficient computation of SVDC, Renderer model is integrated in 3D-HEVC encoder. Block diagram of the Renderer model is shown in Fig. 6.2.

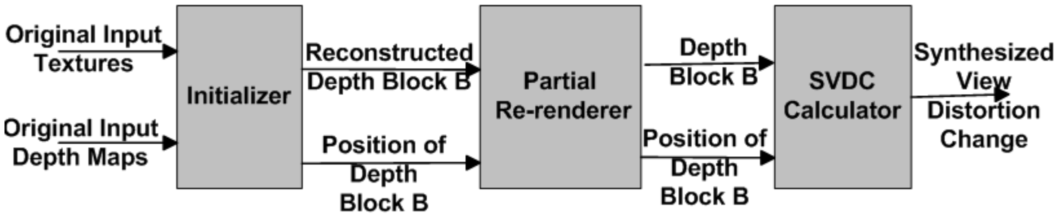

Original Input Textures
Original Input Depth Maps
Initializer
Reconstructed Depth Block B
Position of Depth Block B
Partial Re-renderer
Depth Block B
Position of Depth Block B
SVDC Calculator
Synthesized View Distortion Change

Fig. 6.2 Block Diagram of Renderer Model.

6.2.2 Hardware complexity analysis of Renderer Model

TRenSingleModelC Class basically implements the renderer model for View Synthesis Optimization. This is one of the computational complex part of the 3D-HEVC encoder. Complexity of this class comes from rendering functionalities needed for SVDC computation of each coded depth block. As shown in Fig. 6.2, renderer model consists of three main processing units. Detailed algorithmic description of the renderer model is given [114]. Functional description and hardware complexity analysis of the each part of the renderer is described in the following paragraphs.

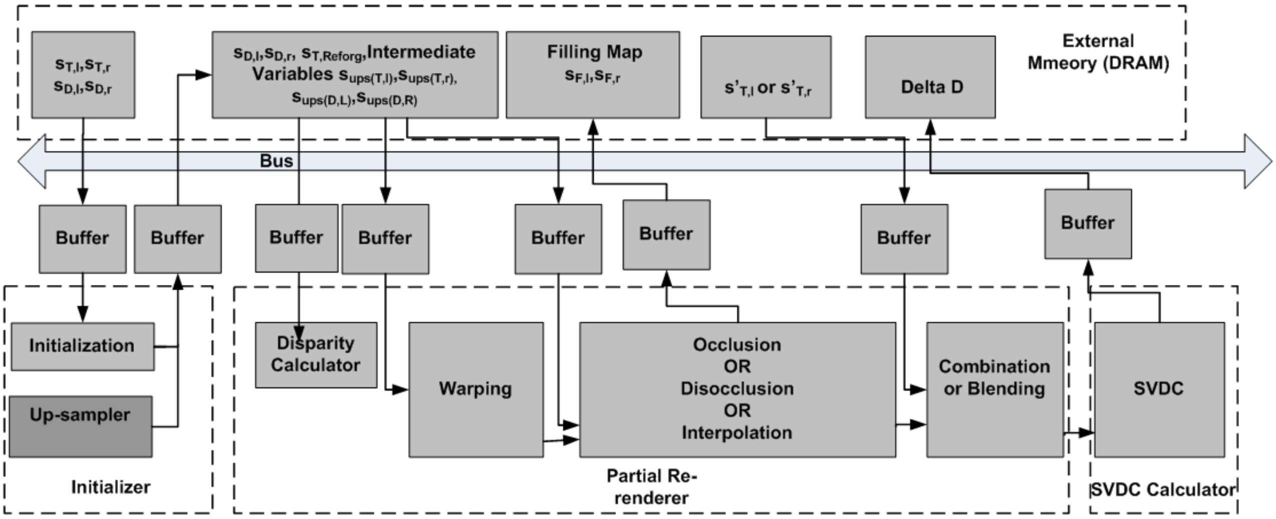


Fig. 6.3 High Level Hardware Architecture of Renderer Model.

Initializer

High level hardware architecture of renderer model is shown in Fig. 6.3. Buffers are used for temporary data storage between the external memory and the processing elements. Initializer synthesize the reference view $S_{T,Reforg}$ from original input textures $S_{T,l}$, $S_{T,r}$ and depth maps $S_{D,l}$, $S_{D,r}$ stored in the T (texture) and depth (D) maps memories, respectively, as shown in Fig. 6.4. Initialization is performed once for every frame. Depth maps are stored as renderer model depth states. Reference view and intermediate variables are stored in the ST (synthesized texture) memory for faster re-rendering. At the time of initialization, the up-sampling of input textures and depth maps is carried out to be used later in interpolation step. Up-sampled textures $S_{ups(T,l)}$, $S_{ups(T,r)}$ and depth maps $S_{ups(D,l)}$, $S_{ups(D,r)}$ are stored in the UpS (Up-sampled) memory.

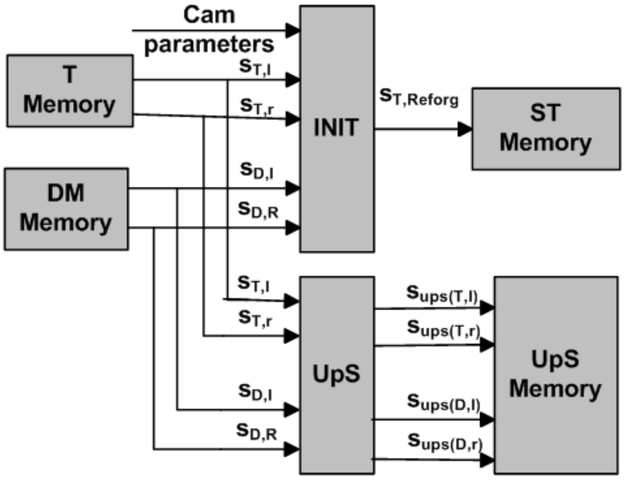


Fig. 6.4 Initializer Hardware Diagram.

Partial Re-renderer

The partial re-rendering of the synthesized view need to be performed when the coding of a depth block is finished. The reconstructed depth block is given to the renderer model, the renderer model updates the depth maps stored as renderer model depth states in the initialization step. Instead of rendering the whole views $S_{T,distCB}$ and $S_{T,orgCB}$ which is computational too complex, the partial re-rendering algorithm is applied which re-renders only parts of synthesized view which got affected by the depth block coding as shown in Fig. 6.5, for right view to left view rendering, the same steps hold for the left view to right view rendering. 1 - 7 are the samples position in the reference and synthesized views and $S_{Disp,r}$ are the disparity values. $D1_{distCB}$ - $D7_{distCB}$ are distortions due to samples 1 - 7, respectively. There are basically different steps involved in the re-renderer algorithm i.e. warping, interpolation, occlusion, dis-occlusion and blending. The hardware diagram of the re-renderer is shown in Fig. 6.6. In the warping the depth maps are warped to the synthesis position by disparity value calculated by

$$d_v = (s * v + o) >> n \tag{6.2}$$

where v, s, o and n represent the depth sample value, transmitted scale factor, transmitted offset vector and shifting parameters, respectively. The depth blocks are stored in the depth memory DM. The warping itself is very complex process requiring the calculation of depth values z from depth maps by the use of camera parameters.

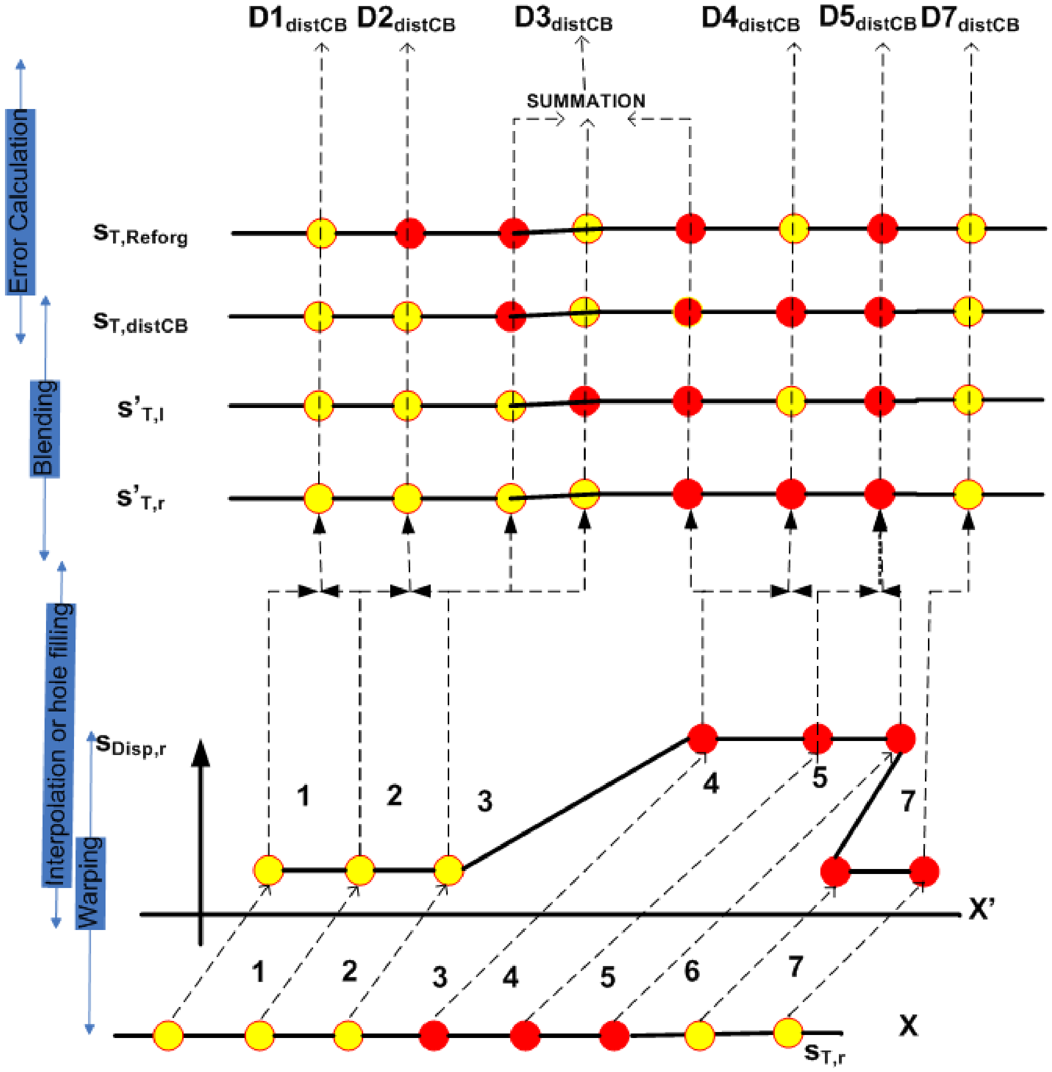


Fig. 6.5 Partial re-rendering algorithm flow diagram.

Warping may cause the original image pixel to be mapped at the fractional pixel position, so rounding required for sub-pixel i.e. half-pel or quarter-pel position depending upon the quality requirement of synthesized view. After the warping, one of the three functions will be carried out i.e. interpolation, occlusion or dis-occlusion. The interpolation position is measured by

$$\widehat{x} = 4\left(\frac{x'_{FP} - x'_s}{x'_e - x'_s} + x_s\right) \tag{6.3}$$

where x'_{FP} is the integer interpolation location between samples interval boundaries x'_s and x'_e of the intermediate synthesized view. $\widehat{x}$ shows the position of the sample value in up-sampled input textures $S_{ups(D,l)}$ or $S_{ups(D,r)}$. Occlusion and dis-occlusion are handled by z-buffering and hole filling algorithms. Filled positions of dis-occluded regions are stored as filling maps $S_{F,l}$, $S_{F,r}$.

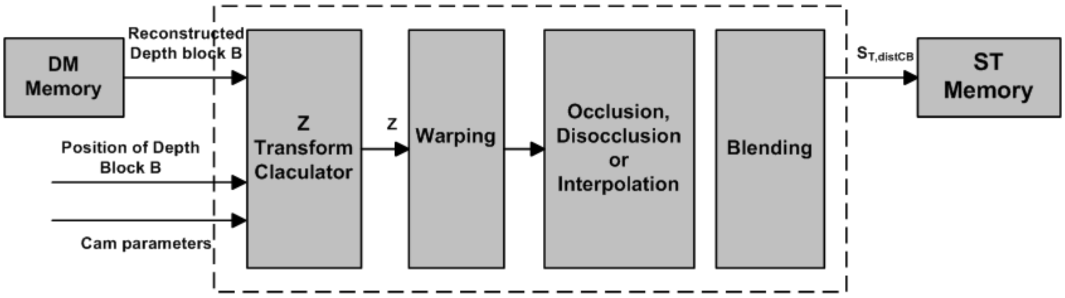

DM Memory
Reconstructed Depth block B
Position of Depth Block B
Cam parameters
Z Transform Claculator
z
Warping
Occlusion, Disocclusion or Interpolation
Blending
ST,distCB
ST Memory

Fig. 6.6 Re-renderer Hardware Diagram.

Blending of synthesized left and right view takes place like blending procedure of view synthesis reference software. Re-rendering process requires high memory cost in terms of z-buffers for occlusion handling, re-order buffers in warping.

SVDC Calculator

The hardware diagram of SVDC calculator is shown in Fig. 6.7. SVDC is calculated by SSD of $S_{T,Reforg}$ and $S_{T,orgCB}$, $S_{T,distCB}$.

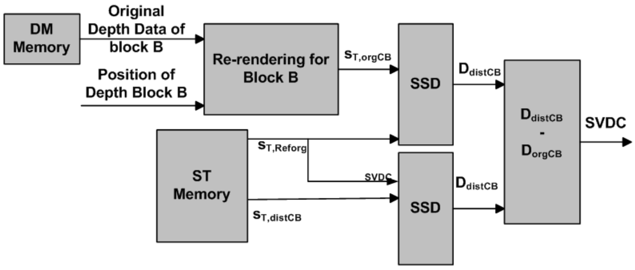


Fig. 6.7 SVDC Calculator Hardware Diagram.

The view synthesized from original texture and depth maps is used as reference view for SVDC calculation. RDO optimization for depth maps coding is carried out i.e. to decide whether to use the applied depth coding mode or not, based on this synthesized view distortion change value.

References

[1] Russell W Burns. *John Logie Baird: Television Pioneer*. Number 28. Iet, 2000.

[2] Randy D Nash and Wai C Wong. Simultaneous transmission of speech and data over an analog channel, April 16 1985. US Patent 4,512,013.

[3] Sara A Bly, Steve R Harrison, and Susan Irwin. Media spaces: bringing people together in a video, audio, and computing environment. *Communications of the ACM*, 36(1):28–46, 1993.

[4] Didier Le Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, 1991.

[5] Gary J Sullivan and Jens-Rainer Ohm. Recent developments in standardization of high efficiency video coding (hevc). In *SPIE Optical Engineering+ Applications*, pages 77980V–77980V. International Society for Optics and Photonics, 2010.

[6] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[7] T Koga. Motion-compensated interframe coding for video conferencing. In *proc. NTC 81*, pages C9–6, 1981.

[8] Fei Sun, Srivaths Ravi, Anand Raghunathan, and Niraj K Jha. Application-specific heterogeneous multiprocessor synthesis using extensible processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(9):1589–1602, 2006.

[9] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[10] Ercan Kalali and Ilker Hamzaoglu. A low energy sub-pixel interpolation hardware. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 1218–1222. IEEE, 2014.

[11] Grzegorz Pastuszak and Mariusz Jakubowski. Optimization of the adaptive computationally-scalable motion estimation and compensation for the hardware h. 264/avc encoder. *Journal of Signal Processing Systems*, 82(3):391–402, 2016.

[12] Carlos Dangelo and Vijay Nagasamy. Specification and design of complex digital systems, June 8 1999. US Patent 5,910,897.

[13] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*, pages 826–831. IEEE Computer Society, 2010.

[14] Daniel D Gajski, Nikil D Dutt, Allen CH Wu, and Steve YL Lin. *High—Level Synthesis: Introduction to Chip and System Design*. Springer Science & Business Media, 2012.

[15] John Polkinghorne and Michael Desnoyers. Application specific integrated circuit, March 28 1989. US Patent 4,816,823.

[16] Shekhar Borkar and Andrew A Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.

[17] Zainalabedin Navabi. *VHDL: Analysis and modeling of digital systems*. McGraw-Hill, Inc., 1997.

[18] Louis J Hafer and Alice C Parker. A formal method for the specification, analysis, and design of register-transfer level digital logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(1):4–18, 1983.

[19] Philippe Coussy and Adam Morawiec. *High-level synthesis: from algorithm to digital circuit*. Springer Science & Business Media, 2008.

[20] Hung-Yi Liu, Michele Petracca, and Luca P Carloni. Compositional system-level design exploration with planning of high-level synthesis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 641–646. EDA Consortium, 2012.

[21] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 13–24. IEEE, 2014.

[22] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170. ACM, 2015.

[23] Benjamin Bross, Woo-Jin Han, Jens-Rainer Ohm, Gary J Sullivan, Ye-Kui Wang, and Thomas Wiegand. High efficiency video coding (hevc) text specification draft 10. *JCTVC-L1003*, 1, 2013.

[24] Joint Video Team. Advanced video coding for generic audiovisual services. *ITU-T Rec. H*, 264:14496–10, 2003.

[25] MPEG-4 Committee et al. Generic coding of moving pictures and associated audio information: Video. *ISO/IEC*, 2000.

[26] Barry G Haskell, Atul Puri, and Arun N Netravali. *Digital video: an introduction to MPEG-2*. Springer Science & Business Media, 1996.

[27] Karel Rijkse. H. 263: Video coding for low-bit-rate communication. *IEEE Communications magazine*, 34(12):42–45, 1996.

[28] Mislav Grgić, Branka Zovko-Cihlar, and Sonja Bauer. Coding of audio-visual objects. In *39th International Symposium Electronics in Marine-ELMAR" 97*, 1997.

[29] Thierry Turletti. *H. 261 software codec for videoconferencing over the Internet*. PhD thesis, INRIA, 1993.

[30] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[31] D Flynn, M Naccari, K Sharman, C Rosewarne, J Sole, GJ Sullivan, and T Suzuki. Hevc range extensions draft 6. *Joint Collaborative Team on Video Coding (JCT-VC) JCTVC-P1005*, pages 9–17, 2014.

[32] J Chen, J Boyce, Y Ye, and MM Hannuksela. Scalable high efficiency video coding draft 3. *Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC N*, 1008, 2014.

[33] Ajay Luthra, Jens-Rainer Ohm, and Jörn Ostermann. Requirements of the scalable enhancement of hevc. *ISO/IEC JTC*, 1, 2012.

[34] Gary Sullivan and Jens-Rainer Ohm. Joint call for proposals on scalable video coding extensions of high efficiency video coding (hevc). *ITU-T Study Group*, 16, 2012.

[35] Ying Chen, Ye-Kui Wang, Kemal Ugur, Miska M Hannuksela, Jani Lainema, and Moncef Gabbouj. The emerging mvc standard for 3d video services. *EURASIP Journal on Applied Signal Processing*, 2009:8, 2009.

[36] Anthony Vetro, Thomas Wiegand, and Gary J Sullivan. Overview of the stereo and multiview video coding extensions of the h. 264/mpeg-4 avc standard. *Proceedings of the IEEE*, 99(4):626–642, 2011.

[37] G Tech, K Wegner, Y Chen, MM Hannuksela, and J Boyce. Mv-hevc draft text 9, document jct3v-i1002. *Sapporo, Japan, Jul*, 2014.

[38] G Tech, K Wegner, Y Chen, and S Yea. 3d-hevc draft text 7, document jct3v-k1001. *Geneva, Switzerland, Feb*, 2015.

[39] Y Chen, G Tech, K Wegner, and S Yea. Test model 11 of 3d-hevc and mv-hevc. *Document of Joint Collaborative Team on 3D Video Coding Extension Development, JCT3V-K1003*, 2015.

[40] H Schwarz, C Bartnik, S Bosse, H Brust, T Hinz, H Lakshman, D Marpe, P Merkle, K Müller, H Rhee, et al. Description of 3d video technology proposal by fraunhofer hhi (hevc compatible; configuration a). *ISO/IEC JTC*, 1, 2011.

[41] Li Zhang, Ying Chen, and Marta Karczewicz. Disparity vector based advanced inter-view prediction in 3d-hevc. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 1632–1635. IEEE, 2013.

[42] Li Zhang, Y Chen, and L Liu. 3d-ce5. h: Merge candidates derivation from disparity vector. *ITU-T SG*, 16, 2012.

[43] L Zhang, Y Chen, X Li, and M Karczewicz. Ce4: Advanced residual prediction for multiview coding. In *Joint Collaborative Team on 3D Video Coding Extensions (JCT-3V) document JCT3V-D0117, 4th Meeting: Incheon, KR*, pages 20–26, 2013.

[44] H Liu, J Jung, J Sung, J Jia, and S Yea. 3d-ce2. h: Results of illumination compensation for inter-view prediction. *ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 JCT3V-B0045*, 2012.

[45] Karsten Muller, Philipp Merkle, Gerhard Tech, and Thomas Wiegand. 3d video coding with depth modeling modes and view synthesis optimization. In *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pages 1–4. IEEE, 2012.

[46] Fabian Jäger. 3d-ce6. h results on simplified depth coding with an optional depth lookup table. *JCT3V-B0036, Shanghai, China*, 2012.

[47] Jin Heo, E Son, and S Yea. 3d-ce6. h: Region boundary chain coding for depth-map. In *Joint Collaborative Team on 3D Video Coding Extensions (JCT-3V) Document JCT3V-A0070, 1st Meeting: Stockholm, Sweden*, pages 16–20, 2012.

[48] YW Chen, JL Lin, YW Huang, and S Lei. 3d-ce3. h results on removal of parsing dependency and picture buffers for motion parameter inheritance. *Joint Collaborative Team on 3D Video Coding Extension Development of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JCT3V-C0137*, 2013.

[49] Sehoon Yea and Anthony Vetro. View synthesis prediction for multiview video coding. *Signal Processing: Image Communication*, 24(1):89–100, 2009.

[50] Christoph Fehn, Eddie Cooke, Oliver Schreer, and Peter Kauff. 3d analysis and image-based rendering for immersive tv applications. *Signal Processing: Image Communication*, 17(9):705–715, 2002.

[51] Gustavo Sanchez, Mário Saldanha, Gabriel Balota, Bruno Zatt, Marcelo Porto, and Luciano Agostini. Dmmfast: a complexity reduction scheme for three-dimensional high-efficiency video coding intraframe depth map coding. *Journal of Electronic Imaging*, 24(2):023011–023011, 2015.

[52] Pin-Chen Kuo, Kuan-Hsing Lu, Yun-Ning Hsu, Bin-Da Liu, and Jar-Ferr Yang. Fast three-dimensional video coding encoding algorithms based on edge information of depth map. *IET Image Processing*, 9(7):587–595, 2015.

[53] Heiko Schwarz and Thomas Wiegand. Inter-view prediction of motion data in multiview video coding. In *Picture Coding Symposium (PCS), 2012*, pages 101–104. IEEE, 2012.

[54] Xiang Li, Li Zhang, and C Ying. Advanced residual predction in 3d-hevc. In *2013 IEEE International Conference on Image Processing*, pages 1747–1751. IEEE, 2013.

[55] Philipp Merkle, Christian Bartnik, Karsten Müller, Detlev Marpe, and Thomas Wiegand. 3d video: Depth coding based on inter-component prediction of block partitions. In *Picture Coding Symposium (PCS), 2012*, pages 149–152. IEEE, 2012.

[56] Martin Winken, Heiko Schwarz, and Thomas Wiegand. Motion vector inheritance for high efficiency 3d video plus depth coding. In *Picture Coding Symposium (PCS), 2012*, pages 53–56. IEEE, 2012.

[57] Gerhard Tech, Heiko Schwarz, Karsten Müller, and Thomas Wiegand. 3d video coding using the synthesized view distortion change. In *Picture Coding Symposium (PCS), 2012*, pages 25–28. IEEE, 2012.

[58] Karsten Muller and Anthony Vetro. Common test conditions of 3dv core experiments. In *JCT3V meeting, JCT3VG1100*, 2014.

[59] Frank Bossen, Benjamin Bross, Karsten Suhring, and David Flynn. Hevc complexity and implementation analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1685–1696, 2012.

[60] Razvan Nane, Vlad-Mihai Sima, Bryan Olivier, Roel Meeuws, Yana Yankova, and Koen Bertels. Dwarv 2.0: A cosy-based c-to-vhdl hardware compiler. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 619–622. IEEE, 2012.

[61] Christian Pilato and Fabrizio Ferrandi. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–4. IEEE, 2013.

[62] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H Anderson, Stephen Brown, and Tomasz Czajkowski. Legup: high-level synthesis for fpga-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 33–36. ACM, 2011.

[63] Razvan Nane, Vlad-Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Fabrizio Ferrandi, et al. A survey and evaluation of fpga high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(10):1591–1604, 2016.

[64] Kazutoshi Wakabayashi and Takumi Okamoto. C-based soc design flow and eda tools: An asic and system vendor perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1507–1522, 2000.

[65] Wim Meeus, Kristof Van Beeck, Toon Goedemé, Jan Meel, and Dirk Stroobandt. An overview of today's high-level synthesis tools. *Design Automation for Embedded Systems*, 16(3):31–51, 2012.

[66] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matthew Moe, and R Reed Taylor. Piperench: A reconfigurable architecture and compiler. *Computer*, 33(4):70–77, 2000.

[67] Nikolaos Kavvadias and Kostas Masselos. Automated synthesis of fsmd-based accelerators for hardware compilation. In *Application-Specific Systems, Architectures and Processors (ASAP), 2012 IEEE 23rd International Conference on*, pages 157–160. IEEE, 2012.

[68] Nikhil Subramanian. *A C-to-FPGA solution for accelerating tomographic reconstruction*. PhD thesis, University of Washington, 2009.

[69] Mentor Graphics. Dk design suite: Handel-c to fpga for algorithm design, 2010.

[70] Walid A Najjar, Wim Bohm, Bruce A Draper, Jeff Hammes, Robert Rinker, J Ross Beveridge, Monica Chawathe, and Charles Ross. High-level language abstraction for reconfigurable computing. *Computer*, 36(8):63–69, 2003.

[71] Timothy J Callahan, John R Hauser, and John Wawrzynek. The garp architecture and c compiler. *Computer*, 33(4):62–69, 2000.

[72] Maya B Gokhale and Janice M Stone. Napa c: Compiling for a hybrid risc/fpga architecture. In *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, pages 126–135. IEEE, 1998.

[73] Y Explorations. excite c to rtl behavioral synthesis 4.1 (a). *Y Explorations (YXI)*, 2010.

[74] Jason Villarreal, Adrian Park, Walid Najjar, and Robert Halstead. Designing modular hardware accelerators in c with roccc 2.0. In *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, pages 127–134. IEEE, 2010.

[75] Calypto Design. Catapult: Product family overview, 2014.

[76] Lars E Thon and Robert W Brodersen. C-to-silicon compilation. In *Proc. of CICC*, pages 11–7, 1992.

[77] Sumit Gupta, Nikil Dutt, Rajesh Gupta, and Alexandru Nicolau. Spark: A high-level synthesis framework for applying parallelizing compiler transformations. In *VLSI Design, 2003. Proceedings. 16th International Conference on*, pages 461–466. IEEE, 2003.

[78] Hui-zheng ZHANG and Peng ZHANG. Integrated design of electronic product based on altium designer [j]. *Radio Communications Technology*, 6:019, 2008.

[79] Ivan Augé, Frédéric Pétrot, François Donnet, and Pascal Gomez. Platform-based design from parallel c specifications. *IEEE transactions on computer-aided design of integrated circuits and systems*, 24(12):1811–1826, 2005.

[80] Justin L Tripp, Maya B Gokhale, and Kristopher D Peterson. Trident: From high-level language to hardware circuitry. *Computer*, 40(3), 2007.

[81] Ravikesh Chandra. *Novel Approaches to Automatic Hardware Acceleration of High-Level Software*. PhD thesis, ResearchSpace@ Auckland, 2013.

[82] Erdal Oruklu, Richard Hanley, Semih Aslan, Christophe Desmouliers, Fernando M Vallina, and Jafar Saniie. System-on-chip design using high-level synthesis tools. *Circuits and Systems*, 3(01):1, 2012.

[83] P Banerjee, N Shenoy, A Choudhary, S Hauck, C Bachmann, M Chang, M Haldar, P Joisha, A Jones, A Kanhare, et al. Match: A matlab compiler for configurable computing systems. *IEEE Computer Magazine*, 1999.

[84] Andrew Putnam, Dave Bennett, Eric Dellinger, Jeff Mason, Prasanna Sundararajan, and Susan Eggers. Chimps: A c-level compilation flow for hybrid cpu-fpga architectures. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 173–178. IEEE, 2008.

[85] Kiran Bondalapati, Pedro Diniz, Phillip Duncan, John Granacki, Mary Hall, Rajeev Jain, and Heidi Ziegler. Defacto: A design environment for adaptive computing technology. In *International Parallel Processing Symposium*, pages 570–578. Springer, 1999.

[86] Oliver Pell, Oskar Mencer, Kuen Hung Tsoi, and Wayne Luk. Maximum performance computing with dataflow engines. In *High-Performance Computing Using FPGAs*, pages 747–774. Springer, 2013.

[87] Satnam Singh and David J Greaves. Kiwi: Synthesis of fpga circuits from parallel programs. In *Field-Programmable Custom Computing Machines, 2008. FCCM'08. 16th International Symposium on*, pages 3–12. IEEE, 2008.

[88] Justin L Tripp, Preston A Jackson, and Brad L Hutchings. Sea cucumber: A synthesizing compiler for fpgas. In *International Conference on Field Programmable Logic and Applications*, pages 875–885. Springer, 2002.

[89] CK Cheng and Liang-Jih Chao. Method and apparatus for clock tree solution synthesis based on design constraints, April 2 2002. US Patent 6,367,060.

[90] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[91] Tom Feist. Vivado design suite. *White Paper*, 5, 2012.

[92] Leon Stok. Data path synthesis. *Integration, the VLSI journal*, 18(1):1–71, 1994.

[93] Christian Pilato, Paolo Mantovani, Giuseppe Di Guglielmo, and Luca P Carloni. System-level memory optimization for high-level synthesis of component-based socs. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014 International Conference on*, pages 1–10. IEEE, 2014.

[94] B Ramakrishna Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proceedings of the 27th annual international symposium on Microarchitecture*, pages 63–74. ACM, 1994.

[95] Florent De Dinechin. Multiplication by rational constants. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(2):98–102, 2012.

[96] Martin Kumm, Martin Hardieck, Jens Willkomm, Peter Zipf, and Uwe Meyer-Baese. Multiple constant multiplication with ternary adders. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–8. IEEE, 2013.

[97] Ganesh Lakshminarayana, Anand Raghunathan, and Niraj K Jha. Incorporating speculative execution into scheduling of control-flow intensive behavioral descriptions. In *Proceedings of the 35th annual Design Automation Conference*, pages 108–113. ACM, 1998.

[98] Hongbin Zheng, Qingrui Liu, Junyi Li, Dihu Chen, and Zixin Wang. A gradual scheduling framework for problem size reduction and cross basic block parallelism exploitation in high-level synthesis. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 780–786. IEEE, 2013.

[99] Scott A Mahlke, Richard E Hank, Roger A Bringmann, John C Gyllenhaal, David M Gallagher, and Wen-mei W Hwu. Characterizing the impact of predicated execution on branch prediction. In *Proceedings of the 27th annual international symposium on Microarchitecture*, pages 217–227. ACM, 1994.

[100] Nancy J Warter, Daniel M Lavery, and WW Hwu. The benefit of predicated execution for software pipelining. In *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, volume 1, pages 497–506. IEEE, 1993.

[101] Kemal Ugur, Alexander Alshin, Elena Alshina, Frank Bossen, Woo-Jin Han, Jeong-Hoon Park, and Jani Lainema. Motion compensated prediction and interpolation filter design in h. 265/hevc. *IEEE Journal of Selected Topics in Signal Processing*, 7(6):946–956, 2013.

[102] Bernd Girod. Motion-compensating prediction with fractional-pel accuracy. *IEEE Transactions on Communications*, 41(4):604–612, 1993.

[103] T Wedi. Motion compensation in h. 264/avc. *IEEE Trans. Circuits Syst. Video Technol*, 13(7):577–586, 2003.

[104] Waqar Ahmad, Maurizio Martina, and Guido Masera. Complexity and implementation analysis of synthesized view distortion estimation architecture in 3d high efficiency video coding. In *3D Imaging (IC3D), 2015 International Conference on*, pages 1–8. IEEE, 2015.

[105] Yao Chen, Swathi T Gurumani, Yun Liang, Guofeng Li, Donghui Guo, Kyle Rupnow, and Deming Chen. Fcuda-noc: A scalable and efficient network-on-chip implementation for the cuda-to-fpga flow. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(6):2220–2233, 2016.

[106] Benjamin Carrion Schafer. Enabling high-level synthesis resource sharing design space exploration in fpgas through automatic internal bitwidth adjustments. 2015.

[107] Jialiang Liu, Xinhua Chen, Yibo Fan, and Xiaoyang Zeng. A full-mode fme vlsi architecture based on 8× 8/4× 4 adaptive hadamard transform for qfhd h. 264/avc encoder. In *VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on*, pages 434–439. IEEE, 2011.

[108] Firas Abdul Ghani, Ercan Kalali, and Ilker Hamzaoglu. Fpga implementations of hevc sub-pixel interpolation using high-level synthesis. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–4. IEEE, 2016.

[109] Zhengyan Guo, Dajiang Zhou, and Satoshi Goto. An optimized mc interpolation architecture for hevc. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 1117–1120. IEEE, 2012.

[110] Cláudio Machado Diniz, Muhammad Shafique, Sergio Bampi, and Jorg Henkel. High-throughput interpolation hardware architecture with coarse-grained reconfigurable datapaths for hevc. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pages 2091–2095. IEEE, 2013.

[111] Grzegorz Pastuszak and Maciej Trochimiuk. Architecture design and efficiency evaluation for the high-throughput interpolation in the hevc encoder. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 423–428. IEEE, 2013.

[112] Sotirios Xydis, Gianluca Palermo, Vittorio Zaccaria, and Cristina Silvano. Spirit: Spectral-aware pareto iterative refinement optimization for supervised high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(1):155–159, 2015.

[113] Cadence Design Systems Inc. Silicon realization enables next-generation ic design. *Cadence EDA360 White Paper*, 2010.

[114] Y. Chen, G. Tech, K. Wegner, and S. Yea. Test model 11 of 3d-hevc and mv-hevc. In *JCT3V meeting, JCT3VK1003*, 2015.