AREA EFFICIENT HARDWARE ARCHITECTURES OF INTRA PREDICTION AND SAMPLE ADAPTIVE OFFSET FILTER FOR HEVC ENCODER

Thesis

Submitted in partial fulfilment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

LAKSHMI

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575025

January, 2023

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

DECLARATION

I hereby ***declare*** that the Research Thesis entitled **AREA EFFICIENT HARDWARE ARCHITECTURES OF INTRA PREDICTION AND SAMPLE ADAPTIVE OFFSET FILTER FOR HEVC ENCODER** which is being submitted to the ***National Institute of Technology Karnataka, Surathkal*** in partial fulfilment of the requirements for the award of the Degree of ***Doctor of Philosophy*** in **Department of Electronics and Communication Engineering** is a ***bonafide report of the research work carried out by me***. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

[illegible]
13/01/23
LAKSHMI
Reg. No. 177034/177EC006
Department of Electronics and
Communication Engineering,
NITK, Surathkal.

Place: NITK-Surathkal.
Date: 13.01.2023

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

CERTIFICATE

This is to certify that the Research Thesis entitled **AREA EFFICIENT HARDWARE ARCHITECTURES OF INTRA PREDICTION AND SAMPLE ADAPTIVE OFFSET FILTER FOR HEVC ENCODER** submitted by **LAKSHMI** (Register Number: 177EC006/177034) as the record of the research work carried out by her, is accepted as the *Research Thesis submission* in partial fulfilment of the requirements for the award of degree of **Doctor of Philosophy**.

[illegible] 13/1/2023

Dr. Aparna P.
Research Supervisor
Assistant Professor
Dept. of E&C Engg.
NITK Surathkal - 575025.

[illegible]
13-1-2023

Chairman-DRPC
Dept. of E&C Engg.
NITK Surathkal - 575025.
(Signature with Date and Seal)

HEAD
[illegible]ent
[illegible]hkal
मंगलूर / MANGALORE - 575 025

Acknowledgements

When I began working on this section, I realised the mountain I was going to have to scale. This led me to appreciate the little ways in which people help you to achieve a goal. I would want to throw the spotlight on these people. I hope they don't have the stage fright!

Dr. Aparna P., my research supervisor for her continuous guidance, encouragement and support during the course of this research work. I appreciate her patience, motivation and help in bringing this thesis to the present form.

My colleagues from my internship at Path Partner Technology, Bangalore for their wonderful collaboration. I would particularly like to single out Mr. Vinay M. K. the then Vice President, Path Partner Technology for his support and the opportunities I was given to further my research work.

My Ph.D. advisory committee members, Dr. Rathnamala Rao, and Dr. Kalpana R. for their insightful comments and encouragement throughout my research work. My sincere thanks to Prof. Ashvini Chaturvedi, Head, Dept. E&C Engg, Prof. Laxminidhi T. and Prof. U. Shripathi Acharya, former Heads, E&C Engg. Dept. for their constant support and help. All the faculty members and staff of Dept. E&C Engg., for their assistance, insights, support and guidance. To all my past teachers.

Friends play a major role. Shruti has been a steady companion during my research work. Our free-wheeling coffee conversations were a lot of fun and a terrific way to unwind. Shwetha and Naveeth, with whom I could discuss my work, findings, results, and occasionally go off on tangents not related to research. Thanks to Shareef, Seenu, Kiran and Venkatesh for their assistance and friendship. Discussions on all topics under the sun and video calls with my gang-nincompoops were great stress busters during COVID lockdowns. And to all my former friends.

My mother, Padma, her love, blessings, sacrifices, kindness and support goes a long way and it is hard for me to contain them in this section. My late father Udaya Kumar, I am sure his blessings and love is always with me. Shiva, my brother, without whom my childhood is incomplete.

My aunt Lalitha, whose continual advice, assistance, and insights helped me grow as a person, and my uncle Mr. Mallikarjunaiah for all his support, kindness and generosity.

My family, in-laws, and my grandparents for their love, support and understanding. Jayachitra, who helped me with my houseworks. And coffee for not letting me to stop working.

Ramesh Babu, my husband, he could have had more hot meals on the table, more serenity at home, and fewer of my temper tantrums if only he hadn't encouraged me to pursue further education. He urged me to dream big, and to work harder to realize them. He worked even harder, leading by example and keeping me grounded. My rocking cycling buddy. I simply can't put into words how much I adore and appreciate him.

Last but the best is my son Siddarth. He accepted my non indulgence, and absence during some long working hours and dealt with it all like a grown-up. He learnt to be independent, managed to help me at home and to edit this section :). Diving with Siddarth was always fun and gave me the opportunity to appreciate and experience the natural beauty of the sea, which is at a short distance from my institute. Before I realised it, he is already a teenager! Thanks for being so understanding and my love and best wishes are always with him.

And to ⟨ your name here please ⟩ who is reading this thesis. Thanks!
Seize the day!

Lakshmi
NITK, Surathkal,
June 2022

To

My husband Ramesh, and my son Siddarth.

Without whom, this thesis would have been

completed ten years back.

Abstract

High efficiency video coding (HEVC) was developed to handle the ever-increasing amount of video content by providing significant compression gains. HEVC/H.265, developed by JCT-VC, can compress 4 K and 8 K videos with 50% more efficiency than its predecessor H.264. This bit-rate saving lowers infrastructure costs, making high-resolution and high-quality video transmissions more affordable. HEVC can handle HD content and deliver better compression efficiency because of computationally complex algorithms like complex partitions, more angular predictions in intra prediction, more parallel tools, a new addition to in-loop filters, and other improved coding tools.

The addition of variable sized prediction units (PUs) and 35 directional predictions has improved the compression efficiency while significantly increasing the complexity of the intra prediction in HEVC. An efficient hardware architecture for the intra prediction is proposed in this thesis which produces high throughput to support high definition (HD) video applications. Features such as a compact reusable reference buffer, a dedicated arithmetic unit are included that reduce hardware resources. The entire architecture works as a pipelined unit and generates eight samples per clock cycle in parallel with no data dependency. All of the above improvements could not be fully utilised when the intra prediction engine is combined with its subsequent transform module in the HEVC encoder. As a result, an improved parallel-pipelined intra prediction engine is designed, which will always process and predict samples row-by-row so that they can be directly transform coded. The read-write latency associated with fetching reference samples is reduced by incorporating a better compact reconfigurable reference buffer in the architecture.

The in-loop filter of the HEVC encoder and decoder is made up of the deblocking filter (DF) and the newly incorporated sample adaptive offset (SAO) filter, which improves the subjective quality of the image. In this thesis, an integrated in-loop filter is designed on hardware that can handle high computations by using very less on-chip memory. The in-loop filter produces high throughput, while handling external memory traffic and dependencies to support Ultra HD video applications.

The architectures are designed in Verilog HDL (Hardware Description Language), synthesised, and then implemented on a 28 nm Artix-7 FPGA board with a dual-core ARM Cortex-M1 processor. Xilinx Vivado is used to generate post-implementation reports for analysis. The experimental results show that the proposed designs achieve high throughput while using very little silicon area and have very high hardware efficiency when compared to several other state-of-the-art hardware architectures.

Keywords: HEVC; intra prediction; in-loop filter; sample adaptive filter; deblocking filter; FPGA; hardware architecture; parallel-pipeline.

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
AMD	Advanced Micro Devices
AMP	Asymmetric Motion Partition
AP SoC	All Programmable System-on-Chip
ARM	Advanced RISC Machine
ASIC	Application-Specific Integrated Circuit
AVC	Advanced Video Coding
B	Bi-predictive
BO	Band Offset
CABAC	Context-Adaptive Binary Arithmetic Coding
CB	Coding Block
CODEC	enCOding and DECoding
CTB	Coding Tree Block
CTU	Coding Tree Unit
CU	Coding Unit
DC	Direct Current
DCT	Discrete Cosine Transform
DF	Deblocking Filter
DSP	Digital Signal Processor
DST	Discrete Sine Transform
EO	Edge Offset
FPGA	Field Programmable Gate Array
fps	frames per second
GPP	General-Purpose Processor
GPU	Graphical Processing Unit
HD	High Definition
HDMI	High-Definition Multimedia Interface
HEVC	High Efficiency Video compression
HF	Horizontal Filter
HM	HEVC testModel
HWE	HardWare Efficiency

I	Intra
IC	Integrated Circuit
IEC	International Electrotechnical Commission
IPE	Intra Prediction Processing Element
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JCT-VC	Joint Collaborative Team on Video Coding
K-Map	Karnaugh Map
LCU	Largest Coding Unit
LSB	Least Significant Bit
LUT	Look-Up Table
MC	Motion Compensation
MD	Mode Decision
MPEG	Moving Picture Expert Group
MSB	Most Significant Bit
NTT	Nippon Telegraph and Telephone
OS	Operating System
P	Predictive
PB	Prediction Block
PDA	Parallel Datapaths Architecture
PPA	Parallel Pipelined Architecture
PSNR	Peak Signal to Noise Ratio
PU	Prediction Unit
QP	Quantisation Parameter
RB	Reference Buffer
RD	Rate-Distortion
RISC	Reduced Instruction Set Computer
RQT	Resuidal-Quad Tree
RS	Reference Selection

SAD	Sum of Absolute Differences
SAO	Sample Adaptive offset
SATD	Sum of Absolute Transformed Differences
SAO	Sample Adaptive Offset
SD	Sample Difference
SD Card	Secure Digital Card
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SSD	Sum of Squared Differences
SSIM	Structural SImilarity Index
SSE	Sum of Squared Errors
TB	Transform Block
TU	Transform Unit
TV	TeleVision
UHD	Ultra High Definition
UHDTV	Ultra High Definition TeleVision
VCEG	Video Coding Experts Group
VCL	Video Coding Layer
VF	Vertical Filter
VoD	video on Demand
VVC	Versatile Video Coding
WPP	Wavefront Parallel Processing

Notations

$\gg$	Right shift operation
Δ	Delta function (differnce)
&	Bitwise AND operation
b	Bit depth of the sample
B_S	Boundary Strength
β	Threshold parameter
t_c	clipping parameter
D	Distortion parameter
E	Sum of differences between original and pre SAO sample
J	Rate distortion cost
λ	Lagrange Multiplier
R	Estimated bit cost
k	sample /pixel position
W	Width of the image
H	Height of the image
C	Corner or top-left reference sample
T	Top reference sample
TR	Top-right reference sample
L	Left reference sample
LB	left-bottom reference sample
N_{Block}	PU size
$Ref[i][j]$	Reference sample at location $[i][j]$
$PredBlock[i][j]$	Predicted sample at location $[i][j]$
$CurrentBlock[i][j]$	Original sample at location $[i][j]$
ind	Index of the reference sample
$fracPrt$	Weight of the reference sample

$RefAng$	Reference array
Ang	Angle associated with directional mode
D_{pre}	Distortion between original and pre SAO samples
D_{post}	Distortion between original and post SAO samples
$p(k)$	original samples
$d(k)$	Pre-SAO samples
O	Offset value
A_{max}	Maximum amplitude of the image
N_P	Number of samples per line
N_L	Number of lines in the image
D_{MSE}	mean Squared Error
DC_{Avg}	Average of top and left references in DC prediction

Chapter 1

Introduction

> There will always be signals, they will always need processing, and there will always be new applications, new mathematics and new implementation technologies.
>
> — Dr. Alan V. Oppenheim, Professor

Videos and their applications are in high demand today and will continue to be so in the future (Cisco Systems, Inc (2020)). Security industries, video-based medical applications, broadcasting, and other industries are currently adopting High Definition (HD) and Ultra High Definition (UHD) video applications. The widespread availability of larger, better, and less expensive smartphones, tablets and other display devices have created the ideal environment for wirelessly consuming, producing, and sharing massive amounts of HD video content. In addition, the pandemic outbreak in the years 2019-21 has resulted in an explosion of video applications and a steep increase in video content. According to the Cisco Annual Report (Cisco Systems, Inc (2020)), internet users will increase from 3.9 billion in 2018 to 5.3 billion by 2023, representing an annual compound growth of 6%. In terms of population, it is that 66% of people globally access internet and majority will access video content. It is also estimated that by 2023, two-thirds i.e. 66% of the installed flat-panel TV sets will be UHD. In short, the current high demand for videos and other video-based applications will continue to grow in the coming years, putting significant strain on existing transmission and storage networks, and resulting in significant bandwidth demands.

Experts from the ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving

Pictures Experts Group (MPEG) jointly developed a new video compression technology High-efficiency Video Coding (HEVC)/H.265, that helps manage ever-increasing bandwidth demands while also coping with existing transmission networks. HEVC is designed to compress 4 K and 8 K videos 50% more efficiently than its predecessor, H.264 (Sullivan and Ohm (2010), Sze *et al.* (2014), Wien (Jan. 2015),). As a result, infrastructure costs are reduced, making high-resolution and high-quality video transmissions more affordable. HEVC's ability to handle HD content and deliver better compression efficiency comes at the cost of a computationally complex algorithm that includes complex partitions, more angular predictions in intra prediction, parallel tools, a new addition introduced to in-loop filters, and other improved coding tools (Bossen *et al.* (2012)).

1.1 Motivation for this Research work

Modern homes, media, enterprises, defence, security, space applications, education, and many other sectors use a wide range of applications based on HD and UHD video content. All of these applications and requirements will continue to expand in future which will result in significant bandwidth demands.

HEVC is developed specifically to support high quality videos and provide twice the compression efficiency than the existing standard. Improved efficiency of HEVC makes high-quality content affordable and attractive to both consumers and distributors. Compared to an H.264 encoder, applications like live streaming with a HEVC-enabled encoder uses half the bitrate. Since the bitrate demands are cut in half, more and more higher resolution video applications can be used. Due to the high demand for high-resolution video applications, major global companies such ase Intel, Telestream (Telestream (2020)), Spin Digital (Spin Digital (2020)), Haivision (Haivision (2020)) and ohters are developing HEVC encoders. Major chip developers like Qualcomm, Ateme, MulticoreWare, NTT, DivX, ViXS, eBrisk Video, Altera, Ittiam, NVIDIA NVENC, AMD UVD, VLC, VITEC and others are developing IC chips to support HEVC encoding and decoding. For applications like live streaming services, an average video streamer can often get away with using software encoding, but cannot handle HD content. The throughput of simulations on software is low, especially with increased processing and large amounts of video data to handle. Whereas, executing algorithms on hardware will increase the throughput by processing multiple units in parallel.

Hardware encoders are turnkey devices designed specifically for the fast, efficient, and reliable encoding of video streams in professional applications. Hardware encoders have more processing power and can work with HD video inputs, allowing users to stream higher-quality video at lower bandwidth rates and with less latency (Sullivan *et al.* (2012)).

With so many devices supporting HD and UHD video, it is significant challenge to the researchers to encode and decode these formats on hardware and produce good quality video to the end-user at a lower bit-rate and higher compression ratio using minimum resources and less power.

1.2 Background

The literature covers the evolution of HEVC as well as several state-of-the-art HEVC hardware implementations.

1.2.1 HEVC and hardware implementations of HEVC

The H.264/MPEG-4, also popular as Advanced Video Coding standard (AVC) was developed in May 2003 and is one of the most widely used standards supporting several multimedia applications, including HDTV broadcasting, video on demand (VoD), blu-ray disc storage, video conferencing and more (Wiegand and Sullivan (2007), Spin Digital (2020)). H.264 as explained in Wiegand *et al.* (2003) is a conventional block-based motion-compensated hybrid video standard based on video coding layer (VCL) design. In comparison to its previous standards, H.264 saves about 50% on bit rates. H.264 was able to achieve such savings by introducing enhanced motion-prediction, smaller block size transforms, adaptive in-loop deblocking filter, better entropy methods. However, H.264 was unable to handle 4 K and 8 K video content at high quality and lower bitrates. This led to the development of the new standard HEVC (Sullivan and Ohm (2010)).

HEVC/H.265 is the state-of-the-art video compression standard suitable for real-time HD and UHD workflows (Ultra HD Forum (2021)) as explained in Sullivan *et al.* (2012). HEVC is capable of reducing the bitrate by 50% for the same quality compared to H.264/AVC and has been widely used in the media industry, driving particularly the shift to UHD videos since its inception (Ohm *et al.* (2012), G. J. Sullivan (2021)).

Tsai *et al.* (2014), design a hardware encoder chip using pipelining technique. Three main stages of pipeline are the prediction core, the reconstruction core, and the bit-stream core. The encoder supports UHD video applications using less resources than H.264. HEVC achieves this efficiency through a number of features, including quadtree partitioning, 35 prediction modes in intra prediction module. Another significant addition is the sample adaptive offset (SAO) filter, a loop filtering block designed to smooth out artifacts caused by the aggressive compression applied by HEVC on the encoding side.

HEVC has been implemented on a variety of platforms, including digital signal processors (DSPs) as in (Norkin *et al.* (2012)), general-purpose processors (GPPs) (Bossen *et al.* (2012)), graphic processing units (GPUs) (Wang *et al.* (2016)) application-specific integrated circuits (ASICs) (Ozcan *et al.* (2013)), (Peesapati *et al.* (2017)), (Shen *et al.* (2016)), (Pastuszak and Abramowski (2016)), multi-core processors (Yan *et al.* (2014)), field-programmable gate arrays (FPGAs) as in (Onishi *et al.* (2018)), (Abeydeera *et al.* (2016)), (Ding *et al.* (2019)) to support various high quality video applications.

1.2.1.1 Earlier works on intra prediction

Quadtree partitioning and 35 angular predictions in intra prediction are two of the most important introductions in HEVC (Wien (Jan. 2015)).

A novel method of intra prediction to provide high compression gains is proposed in Lainema and Ugur (2011). The design was created in such a way that it could be effectively implemented in resource-constrained environments while also being applicable in a wide range of applications. There were more angular predictions and block sizes involved, and the design was intelligently designed to cover these cases by operating in parallel. Their method outperformed previous technologies significantly and consistently across different classes of video material, hence was adopted as the directional intra prediction method for the HEVC draft and later successfully included in the HEVC standard. Best mode selection time reduction techniques were proposed by Zhao *et al.* (2011) and this method is partially included in HEVC test model.

In Lainema *et al.* (2012), an overview of the framework of intra coding for HEVC is provided. For the first time, novel features such as quadtree-based variable block-size coding structure, planar prediction, adaptive filtering (both pre and post-prediction stages), and adaptive scanning were included in intra prediction. All of the design tools were used, and the performance was evaluated in their work. This novel intra prediction algorithm achieved a bitrate reduction of 22% on average and up to 36% objectively over H.264. There were also significant improvements in subjective picture quality. In Sullivan *et al.* (2013), methods to decrease the computational complexity of the intra prediction by selecting the decoded neighbouring blocks for prediction and conditions to skip picture smoothing are included. Zhou *et al.* (2012) emphasis on the loseless encoding, which is used in many real-world applications like automotive vision applications.

Wien (Jan. 2015), and Sze *et al.* (2014) presents an in-depth explanation of the tools used in HEVC intra prediction. Objective and subjective analysis, as well as the complexity and implementation details, are thoroughly discussed and presented. Sze *et al.* (2014) discuss and present various stages of the draft, as well as design changes and improvements in their work. Li *et al.* (2011) noticed that 4×4 PUs account for 66% of the total PUs on the decoder side. They proposed a 4×4 intra prediction engine supporting only 17 directional predictions using two parallel datapaths. Sjövall *et al.* (2015) present a high level synthesis (HLS) design flow for an HEVC intra encoder implemented on a SoC-FPGA. HLS design flow helps to speed up the design exploration, verification, and implementation in hardware/software implementation. In Abeydeera *et al.* (2016), a pipelined HEVC decoder to support 4 K applications is proposed. Single cycle reference processing in intra prediction is used to optimise the architecture. In Amish and Bourennane (2016), a multiplexer based intra prediction engine with parallel datapaths is proposed. They equipped one datapath for DC/ planar mode and the other for directional modes to process 4 K @ 24 fps videos. In Amish and Bourennane (2019), they extend their previous work to design a hardware intra prediction technique to support 3D HEVC video applications.

In Atapattu *et al.* (2016), an FPGA based hardware design of an all intra HEVC encoder that encodes HD quality videos @ 30 fps raw video in real time is presented. However, intra prediction architecture is optimised for 4×4 blocks to predict 19 modes in parallel, and tries to reduce latency rather than considering all prediction directions. In Choudhury and Rangababu (2017), three datapaths one each for DC, planar and for angular modes are proposed. One 4×4 PU is processed in one cycle. The parallel design takes up more resources and lack of reference buffer management increases data loading time. Azgin *et al.* (2017) propose a HEVC intra prediction hardware which uses less energy by optimising the intra prediction computations to support 2 K @ 40 fps. In Kalali *et al.* (2018), they propose a low energy consuming intra prediction hardware for PUs of sizes 4 and 8. In Zhang and Lu (2018), a fully parallel intra encoder design with four parallel prediction units to support each PU size is proposed. However, Zhang and Lu (2018), achieve flexibility and throughput using very high hardware resources. Ding *et al.* (2019) proposes a flexible intra encoder engine with 32 parallel reconfigurable processing elements. Fan *et al.* (2019) used 32 parallel processing elements to support all PU sizes and angular modes. Reference buffers are reused to reduce the resource utilisation.

1.2.1.2 Earlier works on in-loop filters

Another new feature in HEVC is the inclusion of a sample adaptive offset (SAO) filter in the in-loop filters, along with a deblocking filter (DF). The in-loop filter aims to combat block partitioning-related issues such as blocking, ringing, colour bias, and blurring (Sze *et al.* (2014)). However, processing in-loop filter demands greater computing power (Bossen *et al.* (2012)). HEVC in-loop filter has been implemented on a variety of platforms, including digital signal processors (DSPs) (Norkin *et al.* (2012)), general-purpose processors (GPPs) (Bossen *et al.* (2012)), application-specific integrated circuits (ASICs) (Ozcan *et al.* (2013)), (Peesapati *et al.* (2017)), (Shen *et al.* (2016)), (Shen *et al.* (2013*b*)), multi-core processors (Yan *et al.* (2014)) etc. Parallelisation (Ozcan *et al.* (2013)), pipeline (Zhu *et al.* (2013*a*)) and mixed parallel-pipelined (Peesapati *et al.* (2017)) techniques are often used to improve system performance and throughput of in-loop filter.

In Fu *et al.* (2011), they present a coding tree unit (CTU) based SAO to achieve low latency. To adapt to SAO parameters in each CTU, a CTU-based design is specified which are then interleaved into the slice data. A CTU-based optimization algorithm used achieves better rate reduction. The SAO design is evaluated using HM 8.0 (HEVC reference software).

A pipelined SAO filter is proposed in Choi and Joo (2015), to improve the efficiency of HEVC encoders. The SAO is implemented and tested on hardware using four different quantisation parameter (QP) values. In Norkin *et al.* (2014) the SAO filter operations are explained in detail and the goal of SAO is to reduce ringing artifacts caused due to larger transform sizes used in HEVC. Distortion caused due to large transforms is reduced by classifying the samples and adding offset, and this is done on CTUs rather than a whole picture frame which also helps to reduce process complexity. Norkin *et al.* (2014) introduce two classifications edge offset (EO) and band offset (BO) methods to classify and add offsets.

Shukla *et al.* (2017) in their work proposes a hardware efficient SAO filter that can be implemented on both FPGA and ASIC platforms. The boundary samples and CTU in the design, are divided into a set of buffers and operated in parallel. In Rediess *et al.* (2014) the SAO filter is implemented on hardware to speed up the process by taking advantage of hardware parallel operations. Edge offset takes care of the dependency on neighbouring samples by storing them in buffers. For better throughput and efficiency Diniz *et al.* (2015) reuses the hardware in the design. Shen *et al*. also suggested a five-stage pipelined design for a combined DF and SAO filter in their works Shen *et al.* (2013*a*) and Shen *et al.* (2016). Park *et al.* (2016) developed an in-loop filter that uses internal buffers to decrease memory overhead and simplifies processing by modifying the codec to lower hardware costs and increase throughput. Srinivasarao *et al*. in Srinivasarao *et al.* (2015) suggested a design to support both H.264 and HEVC standards for decoders on FPGA and ASIC platforms, to achieve long-term hardware sustainability. Peesapati *et al.* (2017) presents a mixed parallel pipeline DF with low latency. Baldev *et al*. in Baldev *et al.* (2018) proposes a five-stage pipelined in-loop filter for better throughput, expanding on the work of Peesapati *et al.* (2017). Liu *et al.* (2017) offers a reconfigurable in-loop filter using reconfigurable processing units to achieve better performance. Kopperundevi *et al.* (2022) has designed an hybrid DF on FPGA and ASIC to support 4 K and 8 K video applications but considered LCUs of sizes 32×32. Baldev *et al.* (2021) have designed an adaptable parallel and pipeline DF for HEVC decoder. Data dependency is reduced to achieve variable throughput. Singhadia *et al.* (2021) proposed an hardware integrated DF and SAO filter to achieve high throughput. However, design process only luma samples and ignores the chroma samples to accelerate the proecssing.

1.3 Research gap

New additions of quadtree partitioning and 35 angular predictions in intra prediction and SAO filter in in-loop filters pose a significant challenge to implement them on hardware. As seen in previous research works, a lot of effort has gone into implementing intra prediction and SAO filters on hardware in order to design an area and power efficient HEVC encoder. It is also clear from the preceding discussions that HEVC encoders are in high demand. AVC and HEVC standards are currently widely used and will continue to be so until the next standard, Versatile Video Coding (VVC/H.266), becomes stable enough to be developed and integrated into devices.

There are numerous uses for a dedicated HEVC hardware, particularly in light of recent trends in on-the-go video consumption. Since area is a constraint in such devices, the implementation described in this work aims to address this issue. There is a lot of room to further optimise the compression mechanisms and implement them on hardware. All of these factors prompted us to develop and design area efficient block-based intra prediction and SAO filter for HEVC encoders on hardware. The intra prediction and the in-loop filter hardware designed thus could be integrated into the encoder to efficiently encode high-definition video streams in laptops, cellphones, or in dedicated streaming devices. The architectures are tested on FPGA. FPGA was chosen for implementation because of its capabilities such as video acceleration, flexibility, reprogramming, and reconfigurability. FPGAs allow for numerous optimizations to speed up video encoding in comparison to traditional software-based encoders.

1.4 Research Objectives

The primary objective of this work is to design area-efficient hardware architectures of intra prediction engine and SAO filter for the HEVC encoder. The following research objectives are proposed based on the literature review.

1. To design and implement an efficient architecture for planar and DC intra predictions on FPGA to support real time HD applications.

2. To design and implement a pipelined-parallel intra prediction architecture on FPGA that can support all angular modes and prediction unit (PU) sizes, as well as real-time HD applications.

3. To design and implement an area efficient discrete cosine transform/discrete sine transform (DCT/DST) based intra prediction architecture on FPGA for

real-time HD applications.

4. To design and implement an efficient and optimised sample adaptive offset (SAO) filter, as well as combine it with the deblocking filter (DF) of the in-loop filter and implement it on FPGA.

1.5 Contribution to the Thesis

The following are the major contributions of this work:

1. Planar and DC intra predictions are performed using two hardware architectures. The design achieves high throughput to support real-time HD video applications. The following are two architectures that have been designed:
   - Parallel Pipelined Architecture (PPA)
   - Parallel Datapaths Architecture (PDA)
2. An intra prediction engine with the following features is designed to support all of the directional predictions and PU sizes.
   - Reconfigurable reference buffers
   - Dedicated arithmetic unit
   - Reusable multipliers
3. An intra prediction architecture using balanced pipeline, parallel, and sequential techniques is designed. The design employs the following techniques to maximise hardware efficiency and achieve high throughput:
   - DCT/DST based intra prediction engine
   - Reconfigurable reference buffers
   - Dedicated arithmetic unit and reusable multipliers
4. A parallel-pipipelined SAO architecture is designed and integrated with the DF. The optimised design is implemented on FPGA and supports HD and UHD video applications.

1.6 Organisation of the Thesis

A brief chapter-by-chapter organisation of the research thesis is presented in this section.

- **Chapter 1** presents a brief discussion of the motivation for choosing this research topic, followed by a systematic examination of the evolution of the HEVC standard and its implementations. New and improved tools included in intra prediction and SAO filter are presented, along with a brief discussion of their hardware implementations.

- **Chapter 2** covers a general overview, and the evolution of video compression standards. The HEVC standard, its background and coding tools are discussed in detail. This chapter covers the hardware specifications, the test set up, metrics used for validating the proposed algorithms, and the test dataset.

- In **Chapter 3**, objectives 1 and 2 are explained. This chapter covers the optimised hardware architectures for DC and planar predictions, as well as their implementation and analysis. Two architectures with different strategies is implemented and analysed. The best strategy is picked and applied to all the angular predictions of intra prediction module. In this chapter, objective 2, which is an area-efficient intra prediction engine with novel methods to achieve high efficiency is presented. The experimental results are tabulated and analysed. The findings are compared with other similar studies and the results are discussed.

- Objective 3 is presented in **Chapter 4**. A DCT/DST based intra engine designed to work in congruence with the transform module to improve the overall throughput of the HEVC encoder is presented. The design supports all the angular predictions and PU blocks specified in the HEVC codec. The experimental results and observations are tabulated and compared with similar works available in the literature which clearly presents better optimised architecture in terms of area for the throughput obtained.

- **Chapter 5** presents objective 4, in which a hardware implementation of an optimised efficient SAO filter of the in-loop filter is presented. The SAO is designed to work independently and in conjunction with DF of in-loop filter.

The SAO is tested separately and also combined with the deblocking filter. The simulation and implementation results are presented and discussed.

- Finally in **Chapter 6**, the contributions of the proposed architectures and the concluding remarks of this thesis are discussed. Future scope of the proposed design, commercial applications and potential methods for improving the design are discussed.

Chapter 2

HEVC - Overview and key features

Study the science of art. Study the art of science. Develop your senses - especially learn how to see. Realize that everything connects to everything else.

— Leonardo da Vinci, Artist and Scientist

In this chapter, the importance of video coding, the development of video standards and a general overview of the HEVC coding tool is presented. This chapter provides overview of the hardware and metrics used in the evaluation of this study.

2.1 Basics of video compresion

Video is essentially the technology that captures moving images electronically. A video is composed of a series of still images that change so quickly that they create the illusion of a moving image. A higher number of still images/frames results in a smoother motion of a video scene and each frame is recorded as millions of tiny picture elements, called pixels/samples. The number of pixels in an image is referred to as its resolution, and it contains information such as depth, illumination, and texture. Higher frame rates and resolution provides a better video experience, but they also require more samples to be captured and stored, resulting in a storage and transmission bottleneck, necessitating video compression.

The importance of compression is demonstrated using the following example. Consider a 60-minute 1080p raw HD video with an aspect ratio of 16:9 and a frame rate of 30 frames per second (fps). The calculation for storing this video is as follows:

$$\begin{aligned}\text{Storage space required} &= \frac{(W \times H \times bitdepth \times framerate \times \text{video time in seconds})}{8bits/byte} \\ &= \frac{(1920 \times 1080 \times 24 \times 30 \times 60 \times 60 \times) \times 1byte}{8bits} \\ &= 671GB\end{aligned}$$

The video content in the example above requires 671 GB of storage space. And if we assume it costs Rs.40 to deliver this content to a single home. The cost of sending this information to 10,00,000 homes is Rs.40,00,000. Now, if the above video could be compressed by 50%, the same data could be stored or transmitted for half the price, or more data could be stored and transmitted for the same price.

Video compression reduces the size of a large, raw video file into smaller files, and became popular as a way to get more out of fewer resources. However, simply compressing content does not result in the highest possible quality, nor does it allow for the support of multiple devices and platforms. Here, video encoding is used to compress and prepare video files for playback by converting them to the appropriate formats and specifications. Video codecs are hardware or software that compresses and encodes (code and compress) videos for high-quality delivery, then decode/decompresses them for playback. Codec is the short form for enCOding (coding) and DECoding. The final goal of any video codec is to reduce file size and the required bit rate while maintaining the quality of the original source. Figure 2.1, as described in Wang *et al.*

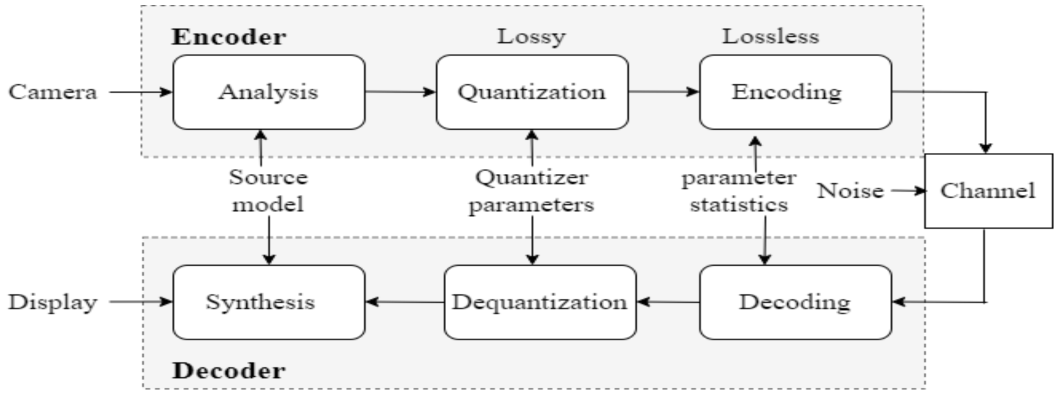


Figure 2.1: Generalized block diagram of the video coding system. (Source: Wang *et al.* (2002))

(2002), illustrates the general operations involved in a video codec. The encoder takes the input video data and converts it to the parameters specified by the codec. These parameters are quantized before being mapped into coded bitstreams and sent over the channel. Reverse encoding is used at the decoder, where the coded bit stream is converted into parameters and dequantized. The video frame is rebuilt by the synthesis tool, and the video is displayed.

Today's video consumption devices include everything from large, high-resolution televisions to powerful palm and pocket-sized viewing devices. Modern video consumption is fuelled by spectacular viewing devices which leads to a never-ending search for more efficient file formats capable of delivering a wide range of high-quality video services.

2.1.1 History and evaluation of video codecs

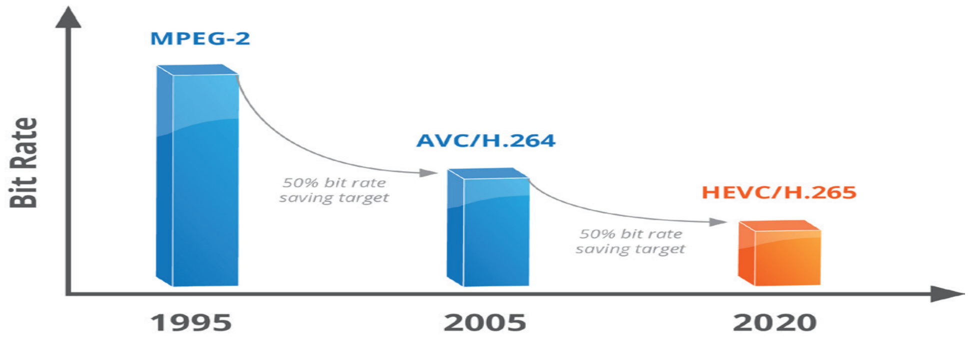


Figure 2.2: Evolution of video codecs with each generation providing better bit rate reduction (Source: Telestream, 2020)

The video coding experts group (VCEG) was formed by the two gaint organisations ISO/IEC and ITU-T, both pioneers in the process of video coding standardisation. Meanwhile, another group of experts from ISO and IEC created the Moving Pictures Experts Group (MPEG), who were responsible for developing standards for compression and transmission of media content. Figure 2.2 shows the evolution of standard codec in the recent years. MPEG-1 gave way to MPEG-2, which became the standard to deliver digital video content using direct broadcast satellite (DBS) for viewing over cable. In the next decade, MPEG-2 made room for MPEG-4, also known as H.264/ AVC.

2.1.2 Advanced Video Coding (AVC)/H.264/MPEG-4

AVC is a popular and widely used codec that allows for a greater variety of services while maintaining the efficiency and quality required to light up tablets, smartphones, and personal computers (PCs) (Wiegand and Sullivan (2007)). H.264 aided the development of high-definition television tools for current distribution models. Many multimedia applications, such as HDTV broadcasting, Video on Demand (VoD), Blu-ray Disc storage, video conferencing, and so on, use AVC, which was followed by H.265.

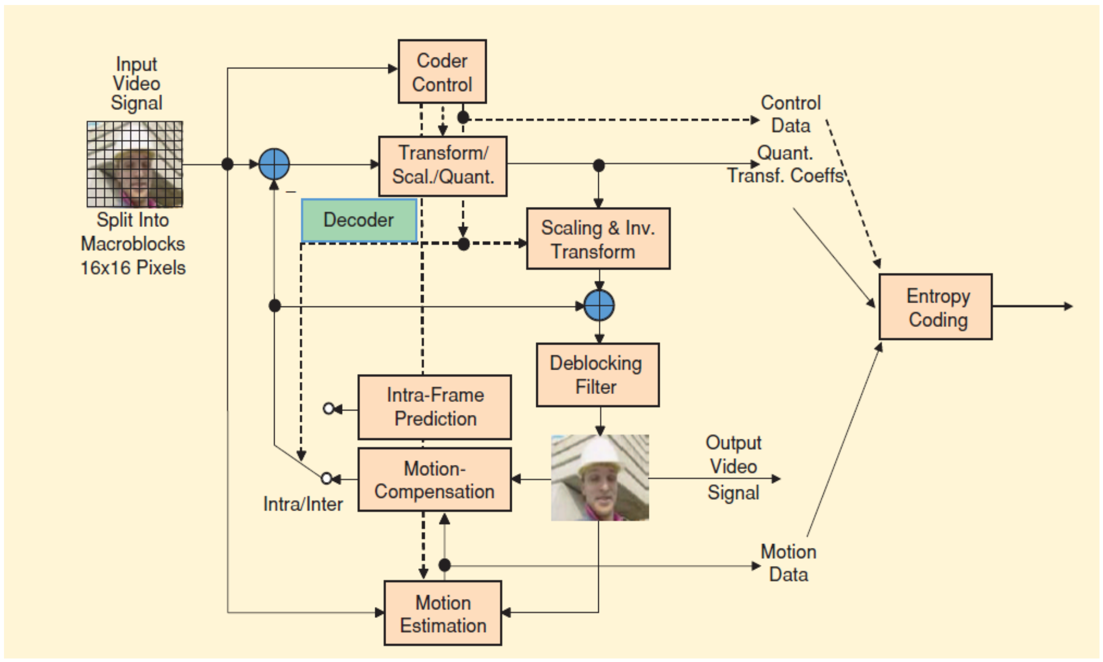


Figure 2.3: Encoding process on a macroblock in AVC (Source: Wiegand *et al.* (2003))

H.264 employs a traditional coding method, namely, hybrid block-based temporal and spatial prediction, followed by block-based scaler quantized block transform coding. Decoding is a subset of encoding processes. A simplified block diagram of the encoding process applied to a macroblock in H.264 is shown in Figure 2.3. The input video frame is divided into squares called macroblocks. Intra tools use spatial details to code the first picture at the start of a sequence or at a random access point. The remaining images in the sequence are coded using motion data and inter prediction tools. The residual difference (the difference between the original and predicted data) is then calculated and transformed using a decorrelating block transform. After scaling and quantizing the transform coefficients, they are encoded and transmitted. The transform coefficients are then inverse scaled and transformed, generating the decoded prediction residual (decoder operation). This parameter is added to the prediction, which is then processed by a deblocking filter, which finally generates the decoded video (Wiegand *et al.* (2003)). AVC supports nine intra coding modes (one DC and eight directional modes) and supports 4×4, 8×8, and 16×16 block sizes. The best prediction mode is the one with the lowest rate-distortion (RD) cost.

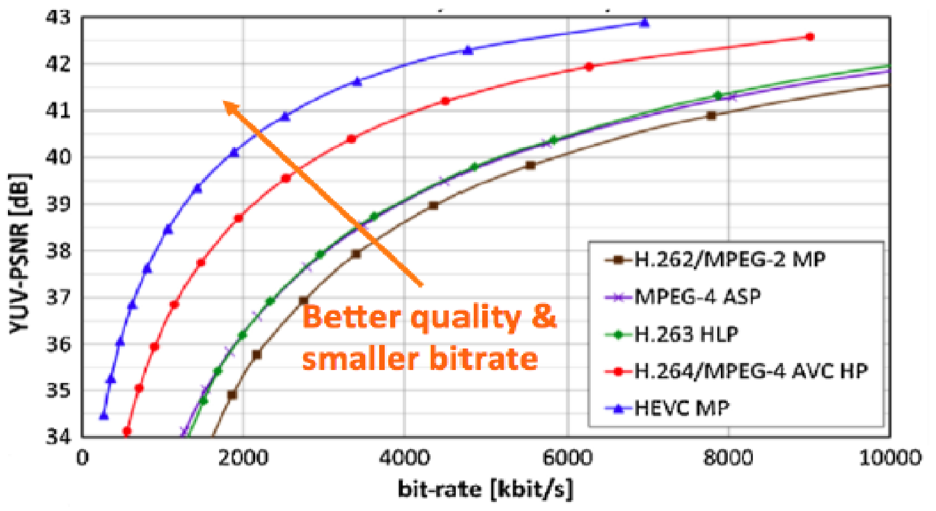


Figure 2.4: Comparison of HEVC with previous standards. HEVC offers better signal to noise ratio for the same bit rate (Source: Sullivan *et al.* (2012))

H.264, on the other hand, has reached the limits of its coding efficiency and is not sufficient for transmitting 8 K video at high quality and low bitrate. HEVC (High Efficiency Video Coding)/H.265 is the most recent ratified standard, which provides significantly higher video quality at the same bit rate or the same video at half the bit rate when compared to H.264. As shown in Figure 2.4, HEVC offers the lowest bit rate for the same quality video content compared to previous codecs (Grois *et al.* (2013), (Sullivan *et al.* (2012))).

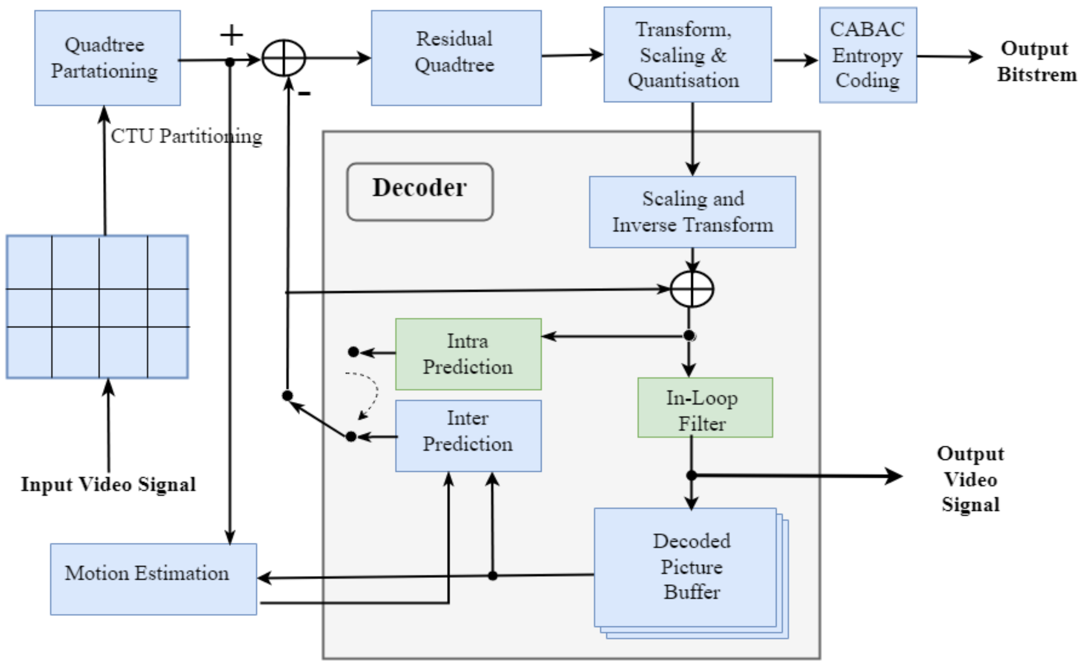


Figure 2.5: Block diagram of HEVC video encoder and decoder.
(Intra prediction module and in-loop filters are highlighted in light green colour, and the decoder is in grey colour)

2.2 High Efficiency Video Coding (HEVC) - An overview

The Video Coding Experts Group (VCEG) of the ITU-T and the ISO/IEC Moving Picture Expert Group collaborated to develop HEVC. By retaining the hybrid coding architecture, HEVC is built on the same general structure as previous standards (Boyce *et al.* (2013)). The block diagram of the HEVC encoder is shown in Figure 2.5. The standard follows initial prediction, either intra frame or inter frame, followed by a transform, quantisation, and entropy coding over the residual information in the standard (Sze *et al.* (2014), Wien (Jan. 2015)). The encoder then performs the decoder operation, which involves inverse scaling and inverse transformation of the transform coefficients, resulting in the decoded prediction residual. This residual is then added to the prediction, which is processed by an in-loop filter, which outputs the decoded video (Sullivan *et al.* (2012)). Despite the fact that the algorithm seems to be similar to the previous standards, there are a few key differences that enable HEVC's enhanced compression capability (Ohm and Sullivan (2012)), which are discussed in the following sections.

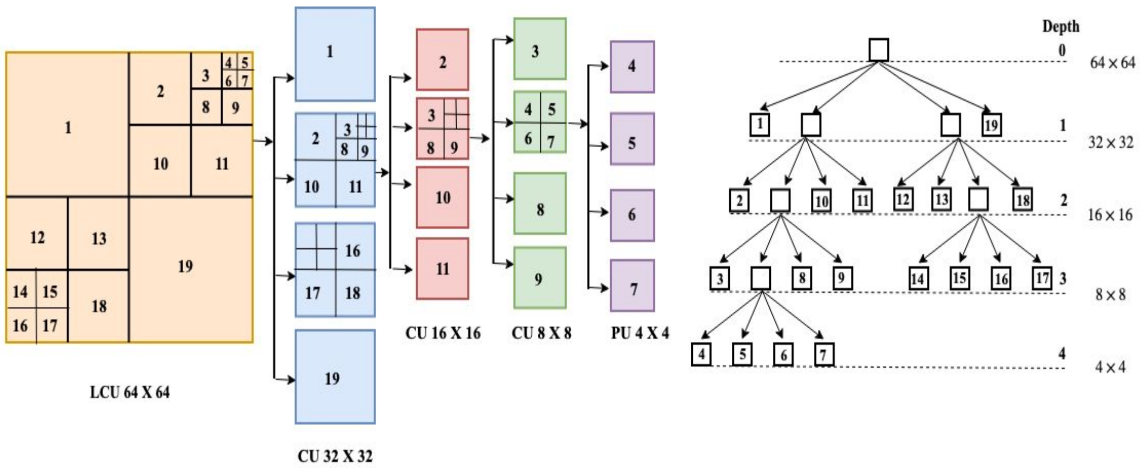


Figure 2.6: CTU is partitioned into CUs in a quad-tree pattern, and CUs are further decomposed into PUs

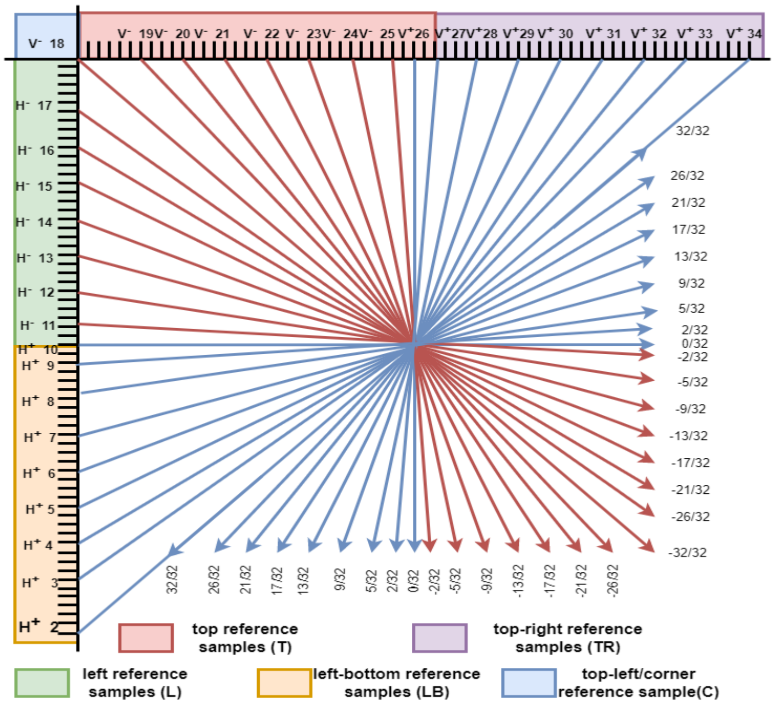

top reference samples (T)
top-right reference samples (TR)
left reference samples (L)
left-bottom reference samples (LB)
top-left/corner reference sample(C)

Figure 2.7: Illustration of reference samples and intra directional modes with their indices used in HEVC (Source: Sullivan *et al.* (2012))

Each input video frame in HEVC is divided into square blocks known as largest coding units (LCUs), which are further divided into coding units (CUs). Each input picture has three colour components: Y (luma), U (chroma), and V (chroma), and each sample in every colour component is represented with 8-bit precision. CUs are further broken down into quad-tree partitioned prediction units (PUs), and transform units (TUs). As shown in Figure 2.6, H.265 introduces PUs of 5 different sizes for intra prediction, namely 4×4, 8×8, 16×16, 32×32 and 64×64 ; and TUs of 4 different sizes, namely 4×4, 8×8, 16×16, and 32×32 (Kim *et al.* (2012), Sze *et al.* (2014)). In contrast, H.264, merely has 4×4 and 16×16 blocks. HEVC uses more complex prediction mechanisms with arithmetic-intensive intra predictions to achieve better compression. As shown in Figure 2.7, HEVC has 35 directional predictions for each type of PU size, whereas H.264 only has 9 modes.

2.2.1 Largest coding unit (LCU)

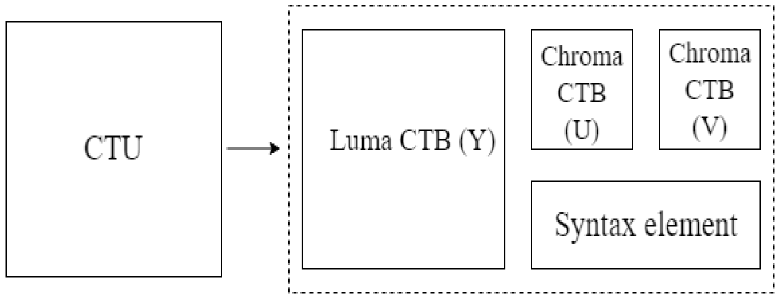


Figure 2.8: Structure of a CTU

CUs are the basic processing units in HEVC. As shown in Figure 2.8, each CTU contains luma coding tree blocks (CTBs), chroma CTBs, and the associated syntax element. The larger LCU size allows for better compression, which is especially beneficial for HD video. CTUs are then divided into CUs and coding blocks (CBs), which are signalled by a quadtree structure, as shown in Figure 2.6, where luma and chroma CUs are usually split together. At the CU level, the decision is made whether to encode a picture area using inter or intra prediction.

2.2.2 Prediction blocks (PBs) and Prediction units (PUs)

The Prediction unit (PU) aggregates the prediction blocks (PBs) of the luma and chroma samples along with the associated syntax elements such as motion data. HEVC supports variable PB sizes ranging from 64×64 down to 4×4 samples. The CUs in I slices can only be in intra-prediction mode, whereas in P and B slices, they can be in both intra and inter prediction mode. HEVC supports both square and rectangular (non-square) partations for inter prediction as shown in Figure 2.9. In Figure 2.9,

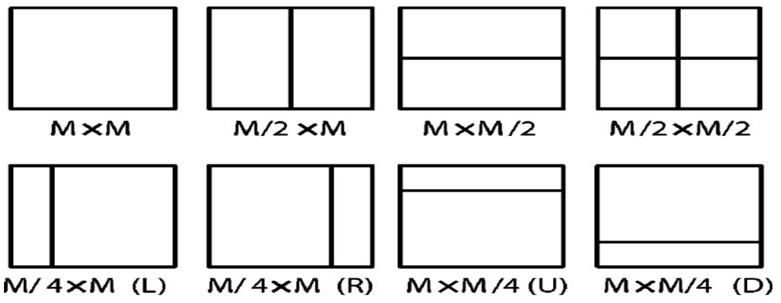


Figure 2.9: PU Partitioning (Source: Sullivan *et al.* (2012))

M denotes the size of the block, where $M = 4, 8, 16, 32$, and L, R, U and D stands for left, right, up and down partition receptively. For intra prediction, however, only square partitions are supported as shown in Figure 2.7.

2.2.3 Transform units (TUs) and Transform blocks (TBs)

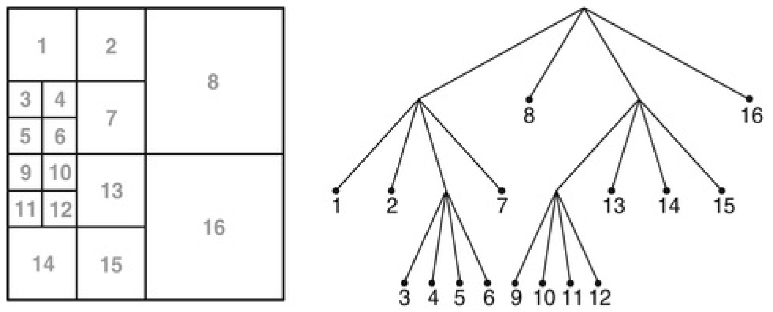


Figure 2.10: Example for the partitioning of a 64×64 CTU into CUs of 8×8 to 32×32, which are further divided into CB and TB samples. (Source: Sze *et al.* (2014))

A transform unit (TU) is a square block of colour component samples on which a two-dimensional transform is used to code the residual signal. The transform unit is made up of luma and chroma transform blocks (TBs), as well as syntax elements that contain the associated transform coffecient levels. CB is partitioned into TBs in a recursive manner using a quadtree approach known as the residual quad-tree (RQT). Three parameters limit each RQT: maximum depth of the tree, minimum allowed transform size and maximum allowed transform size. A TU vary from sizes ranging from 4×4 to 32×32. In order to comply with the transform size constraints, a TB size of 64×64 causes an implicit splitting into four 32×32 TBs. Luma CTBs are subdivided recursively into luma CBs and luma TBs along with the corresponding nested quad-tree structures of the coding tree and residual quad-trees, as shown in

Figure 2.10.

2.2.4 Slices, tiles and wavefront processing

In HEVC input picture frame is split into slices, tiles and wavefront parallel processing (WPP) as shown in the Figure 2.11. Multiple partitions within the image can be created using these techniques, allowing for parallel processing. Slices are a sequence

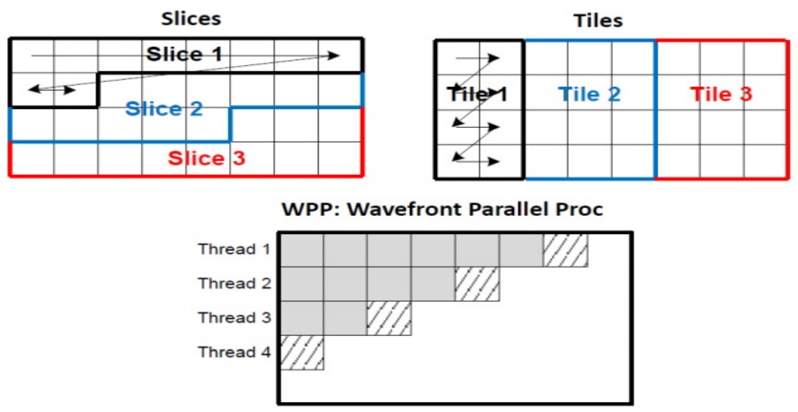


Figure 2.11: Slices, tiles, WPP in HEVC. (Source: esilicon Labs)

of CTUs that are processed in the same order as a raster scan. Each slice can be as big as the entire image or as small as a single CTU. In a picture, there could be one or more slices. As shown in Figure 2.11, a picture can be divided into one or more slices, each of which consists of several non-overlapping slice segments. Within each slice, the first slice segment is always independent.

In addition to slices, HEVC also defines tiles, which are self-contained, independently decodable rectangular regions of the picture. Main purpose of a tile is to allow parallel processing architectures to be used for encoding and decoding. Because multiple tiles are contained in the same slice, they may share header information. A single tile, on the other hand, could contain multiple slices. As shown in Figure 2.11, a tile is a rectangular arranged group of CTUs, with all of them containing about the same number of CTUs (typically, but not necessarily).

In HEVC wavefront parallel processing (WPP), synchronous entropy decoding of multiple CTU rows within a slice is followed. When WPP is enabled, a slice is divided into rows of CTUs. The first row is processed normally, the second row can begin only after two CTUs in the first row have been processed, the third row can begin only after two CTUs have been processed in the second row, and so on. WPP thus provides a finer level of granularity within a slice, allowing for parallel processing.

2.2.5 Inter prediction

In the temporal domain, redundancy refers to the fact that successive frames in time order are usually highly correlated, resulting in parts of the scene being repeated with little or no changes over time. Temporal redundancy, also known as inter-frame correlation. It is clear that coding only the changes in the video content, rather than coding each entire picture, is a more efficient way to represent the video. As stated by Sullivan *et al.* (2012), this technique is called interframe prediction and is designed to improve coding efficiency in order to achieve better compression. As shown in Figure 2.9, inter prediction supports eight PU block partitioning schemes, namely symmetric and asymmetric motion partitions (AMPs). AMP increases coding efficiency by allowing PUs to precisely conform to the shape of objects in the image without the need for additional splitting.

2.2.6 Transforms and quantization

HEVC defines transforms on TUs of size 4×4, 8×8, 16×16, and 32×32, and are simple fixed-point matrix multiplications for the vertical and horizontal components of the inverse transform, as given by the relations (2.1) and (2.2) respectively.

$$Y = s\left(C^T.T\right) \tag{2.1}$$

$$R = Y^T.T \tag{2.2}$$

Where $s()$ is a scaling and saturating function that ensures values of Y. Each factor in the transform matrix T is represented using signed 8 bit numbers. Operations are defined such that 16 bit signed coefficients C are multiplied with the factors, as the transforms are integer approximations of a DCT. Some key modules, such as transforms, intra picture prediction, and motion compensation, are more complex in HEVC than in the previous standard H.264, but the tool attempts to reduce complexity in others, such as entropy coding and deblocking (Bossen *et al.* (2012)).

2.2.7 Entropy coding

The final stage of video encoding, after the video has been reduced to a series of syntax elements, is entropy coding, which is a type of lossless compression. Context adaptive binary arithmetic coding (CABAC) is the technique used in HEVC, which has several key improvements over the previous standard (Sze *et al.* (2014)). Several tools are added to improve throughput, speed and compression performance, as well as methods to reduce context memory requirements. The operation of CABAC encoding can be grouped into three stages.

1. Binarisation - transforms syntax elements into binary symbols (bins)
2. Context modelling - estimating the probability of binary symbols
3. Arithmetic coding - compressing the size of bins to bits based on the probability.

CABAC provides better coding efficiency because of the arithmetic coding tools and more sophisticated context modelling, which also increase tool complexity.

2.3 Semantics of intra prediction

Intra prediction in HEVC is designed to reduce the spatial data redundancy and includes tools to perform accurate predictions. Quadtree partitioning and 35 angular predictions that are introduced to improve compression efficiency also makes the intra engine computationally complex. The current PU is predicted using the neighbouring samples that are already coded. These predicted samples are called reference samples, and are the boundary samples from the previously predicted blocks. Intra prediction can be broadely divided into following stages:

- Reference samples selection
- Mode dependent filtering
- Prediction

2.3.1 Reference samples selection

The reference samples are selected from above and left of the current PU. One 4×4 PU and the reference samples required to predict a PU are illustrated in Figure 2.12.

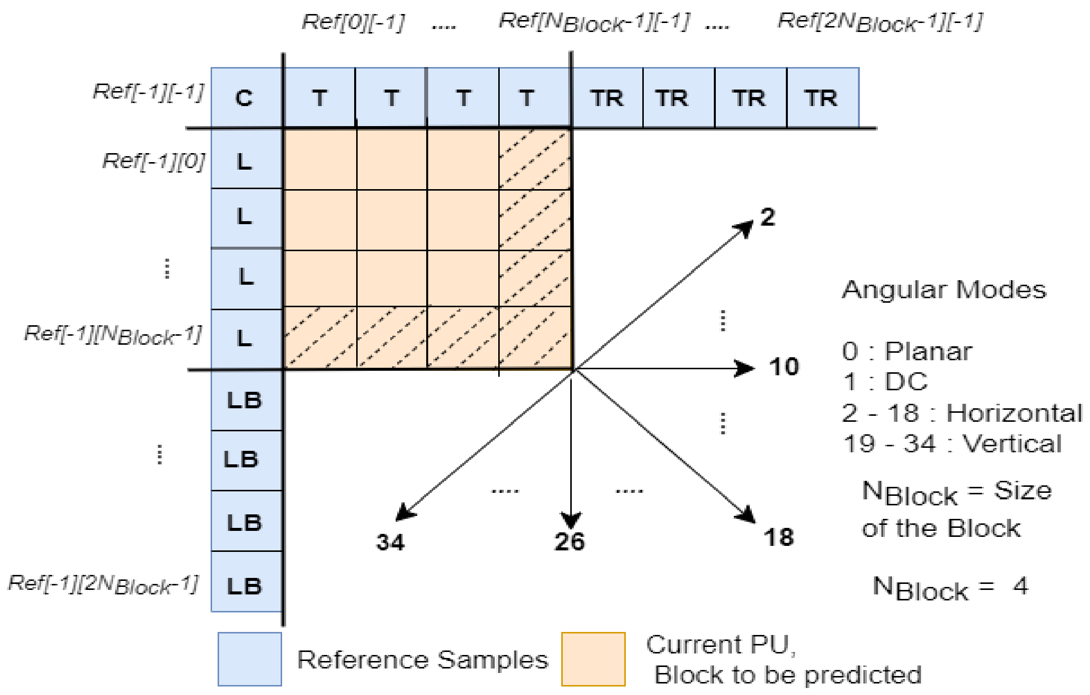


Figure 2.12: Illustration of a PU of size 4 × 4 with the reference samples and angular predictions.

C is the corner or top-left reference sample, T is top, TRs are top-right, $L's$ are left, and LBs are left-bottom reference samples. N_{Block} is the size of the PU under prediction. If any reference samples are missing, the reference array is extended with the last available sample. The reference array is filled with the initial value of 128 (for an 8 bit video) is no reference samples are available. (For example, the first PU in a new input frame has no reference samples). The boundary samples of the predicted PU are used as the references for the following predictions and are highlighted using dashed lines in Figure 2.12. Among the 35 angular predictions introduced, mode 0 and 1 corresponds to planar and DC predictions respectively. $2-35$ modes are angular predictions, with horizontal predictions ranging from $2-18$ and vertical directional predictions ranging from $19-34$ which are shown in Figures 2.7 and 2.12.

2.3.2 Reference samples filtering

Block partitioning of the input frame gives rise to contouring artifacts at the block edges. Smoothing/filtering is applied to reduce contouring artifacts on the reference samples based on the directional modes and PU sizes which are listed in Table 2.1.

Table 2.1: Reference samples smoothing for angular modes based on PU sizes

Block size	Angular modes for which filtering is applied
4×4	No filtering
8×8	2, 18, 34
16×16	2, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 28, 29, 30, 31, 32, 33
32×32	2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33

There are two types of reference sample smoothing:

1. Three-tap filtering
2. Strong intra smoothing.

By default a three tap filter ($[1,2,1]/4$) is applied to each reference sample and its neighboring reference samples. The outermost reference samples are not modified (Sze *et al.* (2014)), but all the other samples are filtered using the relations (2.3), (2.4) and (2.5), where $\gg$ is shift right operation and $[i][j]$ gives the location of the sample.

$$Ref[-1][-1] = (Ref[-1][0] + 2 \times Ref[-1][-1] + Ref[0][-1] + 2) \gg 2 \tag{2.3}$$

$$Ref[-1][j] = (Ref[-1][j+1] + 2 \times Ref[-1][j] + Ref[-1][j-1] + 2) \gg 2 \tag{2.4}$$

$$Ref[i][-1] = (Ref[i+1][-1] + 2 \times Ref[i][-1] + Ref[i-1][-1] + 2) \gg 2 \tag{2.5}$$

$$where,\ i, j = 0, 1, 2, ..., (2N_{Block} - 2).$$

Strong intra smoothing is applied to the samples between $Ref[-1][-1]$ and the corner reference samples. Linear interpolation between the corner reference samples is generated. Strong filtering is applied to PUs of size 32×32 using relations (2.6) and (2.7). Strong filter or three tap filter on a 32×32 PU is decided by the threshold using relations (2.8) and (2.9).

$$Ref[-1][j] = ((63 - j) \times Ref[-1][-1] + (i + 1) \times Ref[-1][63] + 32) \gg 6 \tag{2.6}$$

$$Ref[i][-1] = ((63 - i) \times Ref[-1][-1] + (i + 1) \times Ref[63][-1] + 32) \gg 6 \tag{2.7}$$

$$where,\ i,j = 0,1,2,...,62.$$

$$|Ref[-1][-1] + Ref[2N_{Block}-1][-1] - 2Ref[2N_{Block}-1][-1]| < (1 \ll (b-5)) \quad (2.8)$$

$$|Ref[-1][-1] + Ref[-1][2N_{Block}-1] - 2Ref[-1][2N_{Block}-1]| < (1 \ll (b-5)) \quad (2.9)$$

where b is the bit depth of the sample.

2.3.3 Prediction

For the given angular mode, the samples in a PU are predicted using (2.10) and (2.11), where ind and $fracPrt$ are the index and the weight of the selected reference sample. The reference sample with the index, ind and its offset with index $(ind+1)$ are selected to predict $PredBlock[i][j]$. Mode 2 to 34 are directional prediction modes.

Horizontal predictions is given by:

$$\begin{aligned} PredBlock[i][j] = ((32 - fracPrt) \times RefAng[j + ind + 1] + \\ fractPrt \times RefAng[j + ind + 2] + 16) \gg 5 \end{aligned} \quad (2.10)$$

Vertical predictions is given by:

$$\begin{aligned} PredBlock[i][j] = ((32 - fracPrt) \times RefAng[i + ind + 1] + \\ fracPrt \times RefAng[i + ind + 2] + 16) \gg 5 \end{aligned} \quad (2.11)$$

$$where,\ i,j = 0,1,2,...,(N_{Block}-1).$$

The reconstructed samples are obtained by the addition of the predicted reference samples and the residuals. The residue is the difference between the current and the predicted samples. The reconstructed pixels would become the reference samples for the following predictions.

2.4 In-loop filters

In-loop filtering is applied after the inverse quantization and transformation, but before the reconstructed image is used for predicting other pictures through motion compensation (MC). In-loop filter is a part of both encoder and decoder. The in-loop filter has two tools, which are:

- Deblocking filter
- Sample adaptive offset filter

HEVC, like its predecessor, has an in-loop deblocking filter (DF). DF smooths out blocking artifacts around transform edges in the reconstructed image to improve picture quality and consequently compression efficiency (Norkin *et al.* (2012)). In HEVC, this filter is simple and includes parallel operations. Along with DF, another filtering stage, the sample-adaptive offset (SAO) filter is introduced in HEVC. SAO aims to improve the video quality objectively and subjectively (Fu *et al.* (2012)). In-loop filters corrects distortions introduced during the encoding process (prediction, transform, and quantisation), thereby including filtering as a part of the prediction loop. The pictures thus obtained have less encoding distortion and therefore serve as better references for motion compensated prediction.

2.4.1 Deblocking filter (DF)

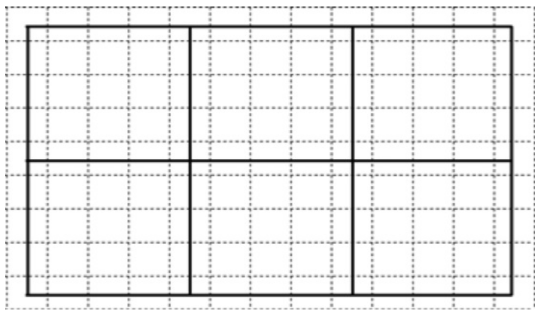


Figure 2.13: Alignment of 8×8 blocks to which the deblocking filter is applied (Source: Bossen *et al.* (2012))

The DF in HEVC uses an adaptive smoothing filter to smooth out the discontinuities that occur between PU and TU block boundaries (Wien (Jan. 2015)). H.264 uses macroblocks in all of its modules, whereas HEVC uses picture-based in-loop filters. A picture is made up of a series of CTUs.

The DF is applied to the horizontal and vertical boundaries of PUs and TUs with a block size of 8×8 while the edges of 4×4 grids are filtered in AVC. This immediately reduces the number of filter modes that must be computed and the number of samples that needs to be filtered by half. In order to enable parallel processing, the order in

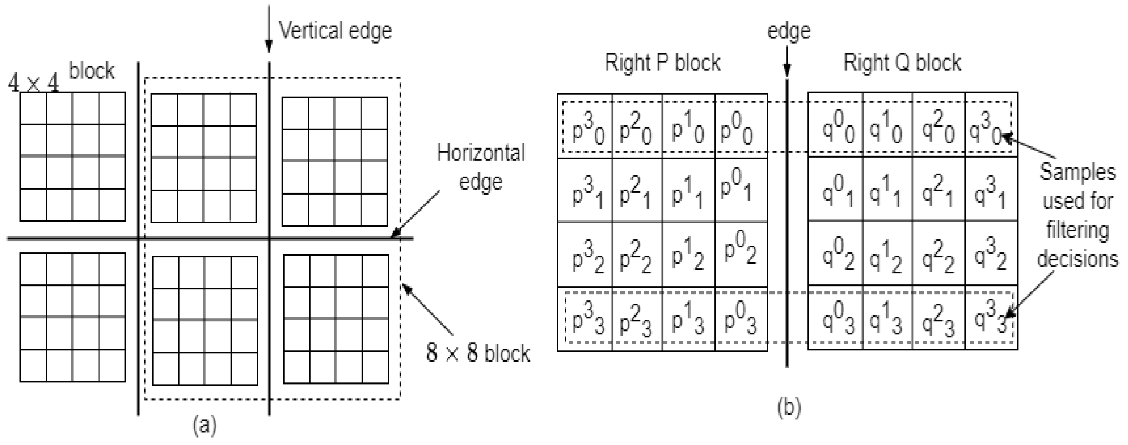


Figure 2.14: Illustration of the block edges in HEVC. (a) Horizontal and vertical edges in HEVC for 8 × 8 blocks. (b) P and Q blocks with the sample convention used for processing

which edges are processed is also changed. Filtering is applied separately on the 4 × 4, P and Q blocks as shown in Figure 2.14. Normal and strong filtering modes are available in DF, which modify two and three luma samples respectively along each boundary. The vertical boundary operation is shown in Figure 2.14 (b), and the horizontal boundary filtering is analogous.

Several decisions must be made before boundary filtering is applied, including whether the boundary should be filtered or not, and which filtering modes, such as normal or strong, should be used. Only data from the first and last rows of P and Q blocks (shown using dotted lines in Figure 2.14 (b)) is used to make filtering decisions. The decisions are made using the relations from (2.12) through (2.21).

$$\begin{gathered}|p_0^2 - 2p_0^1 + p_0^0| + |p_3^2 - 2p_3^1 + p_3^0| + \\ |q_0^2 - 2q_0^1 + q_0^0| + |q_3^2 - 2q_3^1 + q_3^0| < \beta\end{gathered} \tag{2.12}$$

$$|p_0^2 - 2p_0^1 + p_0^0| + |q_0^2 - 2q_0^1 + q_0^0| < \frac{\beta}{8} \tag{2.13}$$

$$|p_3^2 - 2p_3^1 + p_3^0| + |q_3^2 - 2q_3^1 + q_3^0| < \frac{\beta}{8} \tag{2.14}$$

$$|p_0^3 - p_0^0| + |q_0^0 - q_0^3| < \frac{\beta}{8} \tag{2.15}$$

$$|p_3^3 - p_3^0| + |q_3^0 - q_3^3| < \frac{\beta}{8} \tag{2.16}$$

$$|p_0^0 - q_0^0| < 2.5t_c \tag{2.17}$$

$$|p_3^0 - q_3^0| < 2.5t_c \tag{2.18}$$

$$|p_0^2 - 2p_0^1 + p_0^0| + |p_3^2 - 2p_3^1 + p_3^0| < \frac{3\beta}{16} \tag{2.19}$$

$$|q_0^2 - 2q_0^1 + q_0^0| + |q_3^2 - 2q_3^1 + q_3^0| < \frac{3\beta}{16} \tag{2.20}$$

$$|\delta_0| < 10t_c, i = 0, 1, 2, 3 \tag{2.21}$$

The DF avoids filtering real video boundaries, instead focusing on those that are created artificially during the coding phase using filtering decisions (Norkin *et al.* (2012)). Filtering decisions are influenced by the type of block (intra, inter), the quantisation parameter (QP), the motion vector difference, and the video content. A lookup table with QP as the input is used to determine β (threshold) and t_c values.

2.4.1.1 Boundary strength (B_s) Computation

The B_s value which ranges from 0 to 2, determines the amount of filtering required on each block boundary (Norkin *et al.* (2012), Wien (Jan. 2015)). Boundary strength is greatly influenced by the block type (inter/intra). As illustrated in Figure 2.14, edges lying on the 8×8 sample grid are considered, and B_s is calculated for every four samples on either side of the edge using the relation in (2.12), which is also used to compute the t_c (clipping value). The input image is divided into sub-blocks of samples, with edges at the boundaries of 8×8 blocks, as shown in Figure 2.14 (Sze *et al.* (2014)).

2.4.1.2 Filtering decision

The filter operation depends on the β value and skipped, if $B_s > 0$ and if the condition in (2.12) is not met (Boyce *et al.* (2013)). As a result, edges with zero boundary strength are not always filtered because the image may contain very abrupt transitions across the edges which are not related to blocking artifacts. Due to the excessive smoothness, the image becomes blurry when such edges are filtered. Blocking artifacts are more noticeable in low frequency regions of the image. Samples p_i^2, p_i^1, p_i^0, q_i^0, q_i^1, q_i^2 are present on either side of the boundary, as indicated in Figure 2.14(b), and are filtered if they meet the requirements as given by (2.12).

2.4.1.3 Filter implementation

The filtering order is specified as follows, vertical edges are filtered from left to right first, then horizontal edges are filtered from top to bottom. The standard defines two types of filtering:

- Strong filtering mode - modifies three sample values on each side of the boundary
- Standard filtering mode - modifies at most two sample values on each side of the boundary.

2.4.2 Sample adaptive offset (SAO) filter

The SAO filter is a CTU-based filter that reduces degradation by applying an offset to reconstructed images (Fu *et al.* (2011)). When high QP values are used, ringing artifacts appear around the edges of the input image, trying to detract from the subjective quality of the image. The SAO filter reduces ringing effects and thus the difference between restored and original images. Both the luma and chroma components of the image are filtered in the same way with the SAO filter. In SAO, there are two methods for applying offset: edge and band offsets.The encoder can choose to apply either band or edge offset on the different regions of a picture. It can also signal that neither band nor edge offset is used for a region.

2.4.2.1 Edge offset (EO)

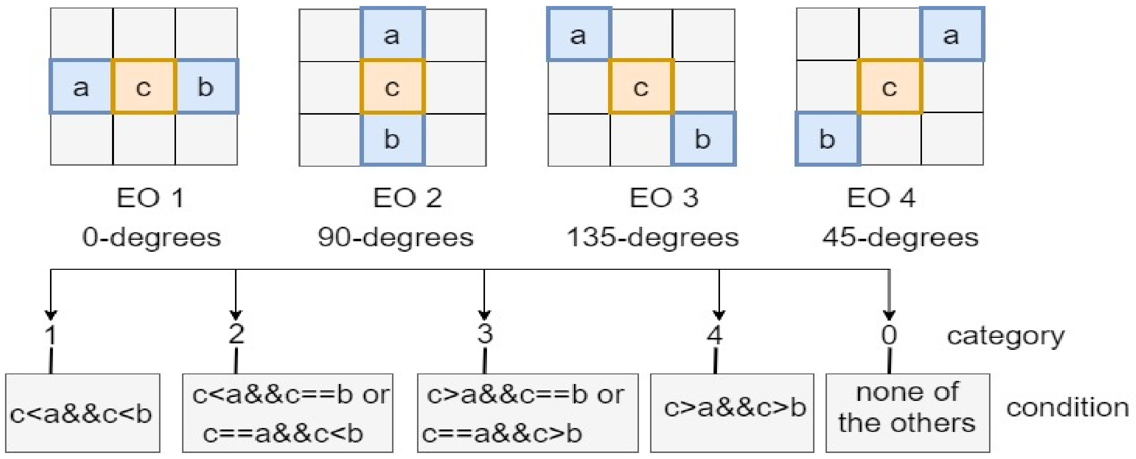


Figure 2.15: Edge offset classes and classification rules

The edge offset in SAO is calculated by comparing the current sample to its two neighbours using edge properties. One of four one-dimensional three-pixel patterns is used to classify samples based on their edge direction and the patterns used are shown in Figure 2.15. Each sample can be classified as a peak (if it is larger than its two neighbours), a valley (if it is smaller than its two neighbours), an edge (if it is the same as its one neighbour), or none of the above. There are four different EO classes, each with five different categories. All angles, categories, and their conditions are shown in Figure 2.15. Altogether there are 20 different offset classes to choose from, and the codec will select the best one. The dependence of the current sample on its neighbouring samples (i.e. samples from the left, right, top, bottom, top left, bottom right, top right, top right, or bottom left CTUs) is one of the challenges in implementing EO, which causes latency. This dependency and latency must be factored into the pipeline techniques in hardware implementation. EO calculations in HEVC requires more operations and is dependent on two neighbouring samples.

2.4.2.2 Band offset (BO)

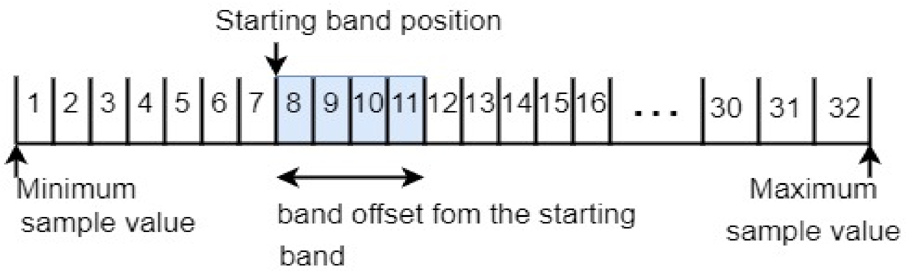


Figure 2.16: Illustration of band offset and offset determination

In the case of band offset, all samples in a CTU are divided into 32 uniform bands from zero to the maximum intensity. For 8-bit data, the maximum value is 255, so the bands are 256/32 = 8 pixels wide. The 32 bands are divided into two groups. The decoder receives just the offset values of four successive bands and the starting band position in HEVC. The band indexes are assigned as 1 - 4. The remaining 28 band offset values and band indexes are set to zero. The band offset and offset determination are depicted in Figure 2.16.

2.4.2.3 Fast distortion estimation

Fast distortion estimation method is used to determine the best offset and SAO type (EO and BO). First step is to add offsets to pre-SAO samples, then generate post-SAO samples and finally calculate distortion between original samples and post-SAO samples, this is given by,

$$D_{pre} = \sum_{k \in C} (p(k) - d(k))^2 \tag{2.22}$$

Where k is pixel position, $p(k)$ is original samples, and $d(k)$ is pre-SAO samples, k belongs to C and C is a set of specified SAO (EO, BO) samples. Next, the distortion between the original samples and the post-SAO samples are generated using the following equation.

$$D_{post} = \sum_{k \in C} (p(k) - (d(k) + O))^2 \tag{2.23}$$

Where O is the offset of a given sample set.

Then the delta distortion, difference between D_{post} and D_{pre} is generated.

$$\begin{aligned} \Delta D &= D_{post} - D_{pre} \\ &= NO^2 - 2OE \end{aligned} \tag{2.24}$$

$$E = \sum_{k \in C} (p(k) - d(k)) \tag{2.25}$$

In (2.24), N stands for number of samples, and E in (2.25) is the sum of differences between original and pre-SAO samples. The following equation is used to calculate the delta rate-distortion (RD) cost:

$$\Delta J = \Delta D + \lambda R \tag{2.26}$$

Where λ is the lagrange multiplier, and R is the estimated bit cost.

ΔJ for all the bands are generated, using this value rate-distortion impact of SAO on the LCU is estimated. This values are used to determine, band transition position, or the applicable edge offset class.

The addition of SAO tool in the standard adds to the complexity, as it may require either an additional decoding pass, or an increase in line buffers. The offsets are transmitted in the bitstream and thus need to be derived by an encoder. If considering all SAO modes, the search process in the encoder can be expected to require about an order of magnitude more computation than the SAO decoding process. Compared to H.264/AVC, where only a deblocking filter is applied in the decoding loop, the current HEVC specification features an additional sample-adaptive offset (SAO) filter. This filter represents an additional stage, thereby increasing complexity.

2.5 Hardware specifications of the FPGA board

Intra prediction and SAO filter modules are designed in Verilog HDL (Hardware Description Language), synthesised, and implemented on a 28nm Artix-7 FPGA board with a dual-core ARM Cortex-M1 processor. Xilinx Vivado is used to generate simulation, pre/post synthesis and pre/post implementation reports for analysis.

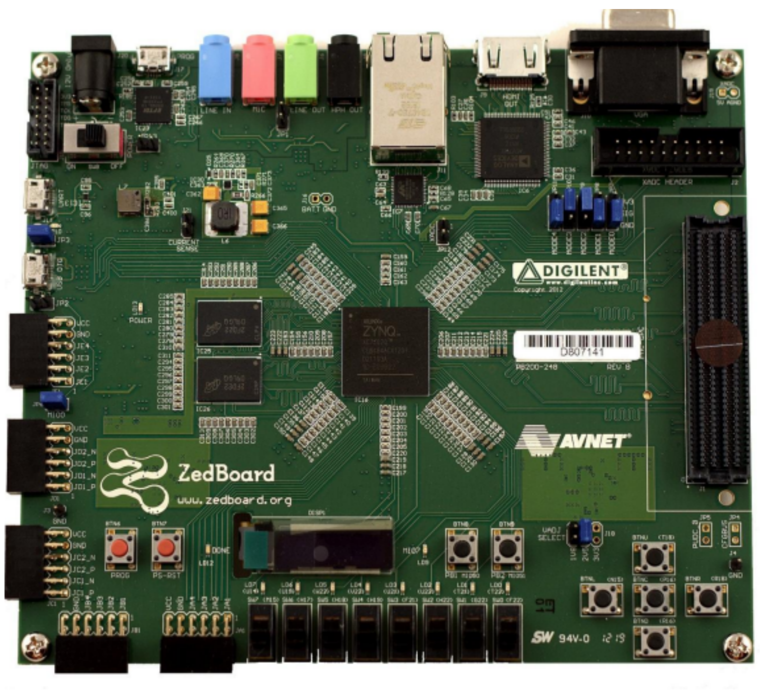


Figure 2.17: Zed board (Source: Guide to Zed (2012))

Hardware architectures are tested on Z-7020 boards. These boards are feature-rich, ready-to-use, entry-level embedded software and digital circuit development platforms built around the smallest member of the Xilinx Zynq-7000 family. The Z-7000 SoC boards (Zed board and Xilinx Artix-7 FPGA AC701 Evaluation Kit shown in Figures

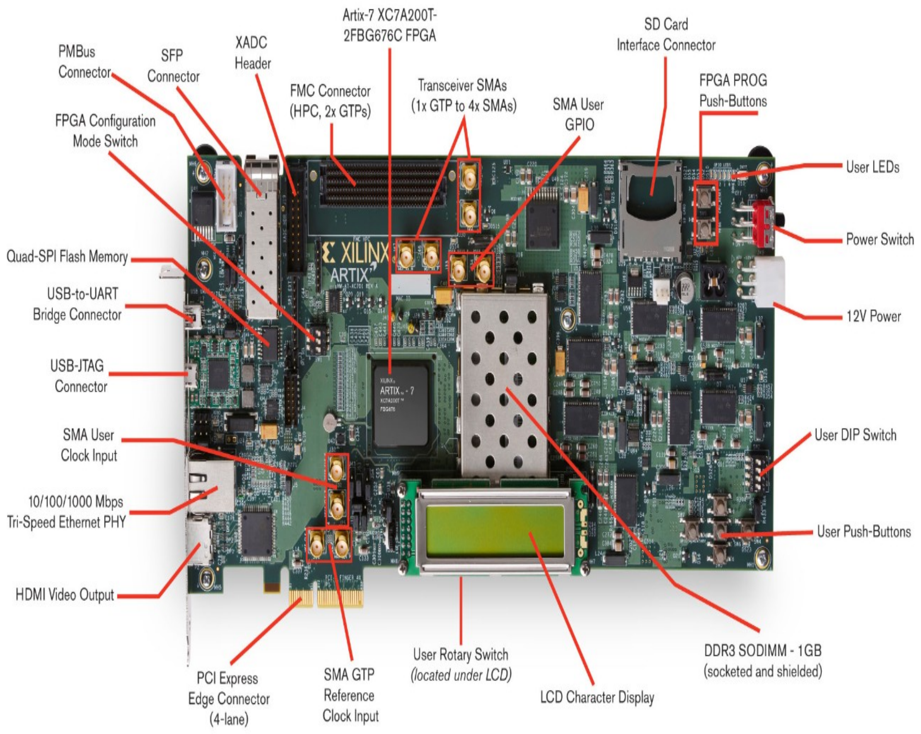

Artix-7 XC7A200T-2FBG676C FPGA
SD Card Interface Connector
PMBus Connector
SFP Connector
XADC Header
FMC Connector (HPC, 2x GTPs)
Transceiver SMAs (1x GTP to 4x SMAs)
SMA User GPIO
FPGA PROG Push-Buttons
FPGA Configuration Mode Switch
User LEDs
Power Switch
Quad-SPI Flash Memory
USB-to-UART Bridge Connector
12V Power
USB-JTAG Connector
SMA User Clock Input
User DIP Switch
10/100/1000 Mbps Tri-Speed Ethernet PHY
User Push-Buttons
HDMI Video Output
DDR3 SODIMM - 1GB (socketed and shielded)
User Rotary Switch (located under LCD)
PCI Express Edge Connector (4-lane)
SMA GTP Reference Clock Input
LCD Character Display

Figure 2.18: Xilinx Artix-7 FPGA AC701 board details (Source: Guide to AC701 (2013))

2.17 and 2.18 respectively) are based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series FPGA logic. The zed board has 85,000 logic cells, 512 MB DDR3, 560 KB RAM, 220 DSP slices and operates at 150 MHz frequency, while the Xilinx AC701 has 2,15,360 logic cells, 1 GB DDR3, 32 MB flash memory, 8 KB EEPROM, 740 DSP slices and operates at 200 MHz (Guide to Zed (2012), Guide to AC701 (2013)). Both boards come with a wide range of multimedia and connectivity peripherals, allowing them to host a complete system design. The on-board memories, video and audio I/O, dual-role USB, Ethernet, and SD slot will help the design up-and-ready with no need of additional hardware.

FPGAs have the advantage that they are reprogrammable and made up of a matrix of programmable logic blocks connected with programmable interconnects. FPGAs are also an attractive option due to their flexibility, hardware timed speed, reliability, and parallelism. FPGAs parallel nature allows each processing task to operate independent of other logic blocks and doesn't compete for resources (NI CORP (2021)).

Current FPGAs are heterogeneous in nature, i.e, they are made up of a microprocessor and then connected to I/O (Input/Output). Modern FPGA's operate with logic gates combined with processors all in a single chip called as SoC (System on Chip), resulting in improved computing performance (AMD Xilinx (2012)). This approach takes advantage of the benefits offered by both the targets. In short, modern FPGAs offer high logic density, an embedded host processor, DSP blocks, clocking, and high speed at a low cost, as well as the ability to add artificial intelligence (AI) to their platform, which is highly desired.

2.6 Test set up for the objective analysis of the hardware designs

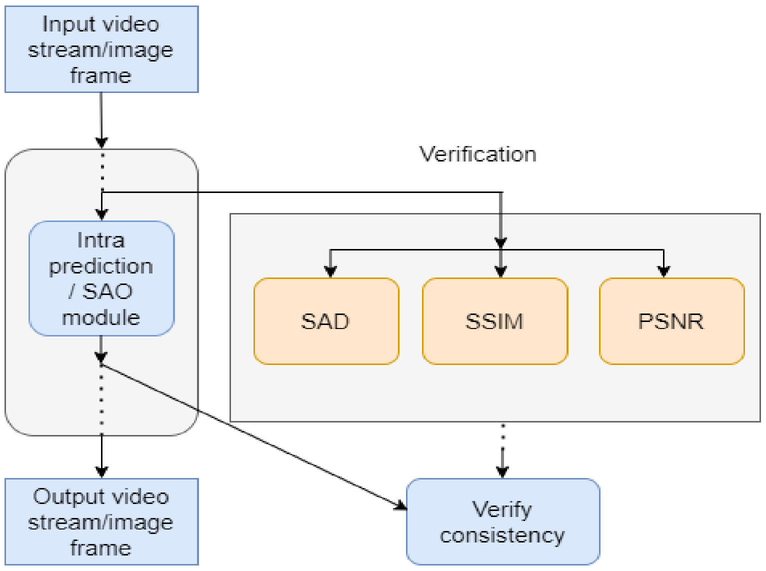

Input video stream/image frame
Verification
Intra prediction / SAO module
SAD
SSIM
PSNR
Output video stream/image frame
Verify consistency

Figure 2.19: Intra and SAO modules test verification chart

One of the biggest challenges of implementing the intra prediction and in-loop modules on FPGA is to correctly verify its behaviour. In order to do this, several steps were taken. Reference software is provided by the JCT-VC, using this reference code, the modules, input and output data required to our design modules are extracted. This data is used as input in the software model to generate output data. This data is used to test the correctness of the implementation of the intra prediction and in-loop modules. The data is also tested using other metrics allowed in the standard. Figure 2.19 shows the test verification chart used to test the architectures, in which the

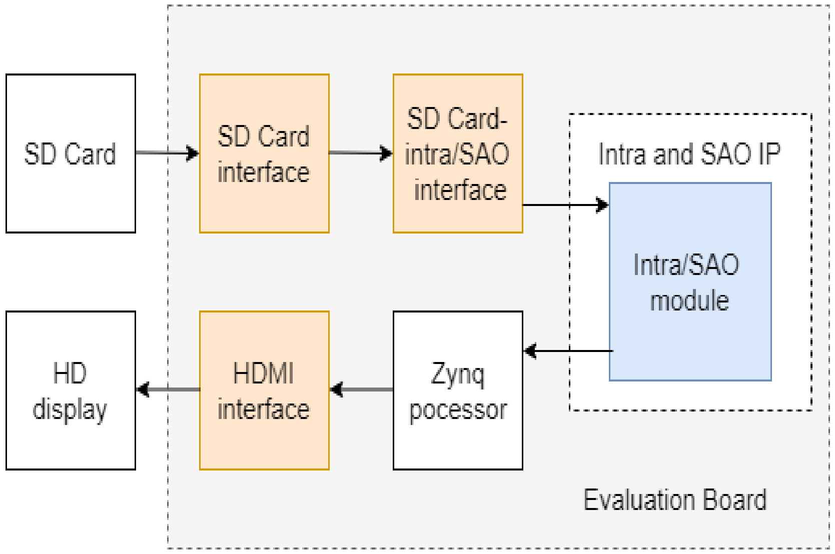


Figure 2.20: Block diagram of Intra and SAO filter IP test setup on the FPGA

video/image frames are fed into the intra/SAO modules and the output is checked for consistency. The hardware model is run using the input data specified by the HEVC reference development team and the output is compared to the reference output to verify that the hardware is doing the processing as expected. The correctness of the output data in the hardware is also tested using the metrics described in the following sections. The demo of the FPGA implementation used to test the architectures is shown in Figure 2.20.

2.7 Metrics used for the objective analysis of the hardware design

The distortion or the correctness of the output data is compared to the original data to determine reconstruction quality. This measurement is frequently performed on a sample-by-sample basis. Sum of Squared Difference (SSD), Sum of Absolute Differences (SAD), Sum of Absolute Transformed Differences (SATD), and PSNR - Peak Signal-to-Noise Ratio - are the distortion measures used in HEVC. To perform software testing, the reference software, HM software test module, is used (Fraunhofer (2015)).

2.7.1 Sum of Absolute Differences (SAD)

The sum of absolute differences is defined as the sum of the difference between the current and the predicted PU and is given by (2.27). SAD is one of the most straightforward and quick metrics available (Wien (2015)).

$$SAD = \sum_{i=0}^{N_{block}} \sum_{j=0}^{N_{block}} |PredBlock[i][j] - CurrentBlock[i][j]| \tag{2.27}$$

2.7.2 Structural similarity index (SSIM)

The SSIM is a perceptual metric that quantifies image quality degradation as a result of processing such as data compression or data transmission losses. SSIM is a full reference metric that requires two images, the original and a processed image, which is a compressed image, from the same image capture. The SSIM scale runs from 0 to 1, with 1 denoting perfect match.

2.7.3 Peak signal-to-noise ratio (PSNR)

The peak signal-to-noise ratio (PSNR) between two images is calculated in decibels. The PSNR is used to determine the quality of the reconstructed image. The higher the PSNR, the better the image quality. PSNR is calculated as shown below.

$$PSNR = 10\log_{10}\left(N_P N_L \frac{A_{max}^2}{D_{MSE}}\right) db \tag{2.28}$$

Where A_{max} is the maximum amplitude of the original image. For a 8-bit video it is $2^8 - 1 = 255$. N_P is the number of samples per line, N_L is the number of lines in the image. D_{MSE} is the mean squared error (MSE). MSE is the average of the squares of the errors between the original and reconstructed images, and error is the difference in the value between the original and processed images.

2.7.4 Metrics used to measure hardware complexity

Hardware encoders are not cross-platform and are designed to run in real-time. They are either built into the platform or are an add-on. One of the most important requirements for hardware encoders is to have area efficient architecture wih low power consumption. To compare different algorithms and hardware implementation results, performance metrics such as area occupied, number of gates, number of registers, memory utilisation, frequency of operation, throughput, power consumption, and board technology are used.

2.8 Test sequences used for the objective analysis of the hardware designs

Table 2.2: Test sequences from the JCT-VC common testing conditions

Class	Sequence Name	Number of Frames	Resolution	Frames per second
A	Traffic	150	2560 × 1600	30
	PeopleOnStreet	150	2560 × 1600	30
	Nebuta	300	2560 × 1600	60
	SteamLocomotive	300	2560 × 1600	60
B	Kimono	240	1920 × 1080	24
	BasketballDrive	500	1920 x 1080	50
C	RaceHorses	300	832 x 480	30
	BasketballDrill	500	832 x 480	30
D	RaceHorses	300	416 x 240	30
	BlowingBubbles	500	416 x 240	50
E	FourPeople	600	1280 x 720	60
F	BasketballDrillText	500	832 x 480	50
	SlideShow	500	1280 x 720	20
	ChinaSpeed	500	1024 x 768	30

(Source:Sze *et al.* (2014))

The test sequences that have been used during the development of HEVC are listed in Table 2.2. Video sequences of 8-bit depth are used to test the hardware architectures. All sequences but two of the sequences in class A have a bit depth of 8 bits. The class A sequences are cropped regions of 2560 × 1600 samples from ultra high resolution sequences (Traffic: 3840 × 2160 - 8 bit, PeopleOnStreet: 3840 × 2048 - 8 bit, Nebuta and Steam Locomotive: 7680 × 4320 - 10 bit).

2.9 Summary

The importance and development of video standards along with a general overview of the HEVC coding tool is presented. Intra prediction and integrated in-loop filter is discussed in more detail. A detailed overview of the hardware used in this work and evaluation metrics and methods used in this study are presented.

Chapter 3

Design and implementation of area-efficient intra prediction engine for HEVC encoder

> If you keep proving stuff that others have done, getting confidence, increasing the complexities of your solutions – for the fun of it – then one day you will turn around and discover that nobody actually did that one!
>
> — Richard P. Feynman, Physicist

3.1 Introduction

In this chapter, a new, area efficient mixed parallel-pipelined intra prediction architecture on hardware is proposed.
The highlights of this chapter are summarised as follows:

1. An efficient hardware architecture having a high throughput to support the real time HD applications is designed for DC and planar intra predictions on FPGA. Two different hardware architectures are designed and their performances are compared.

2. An area efficient mixed pipelined-parallel intra prediction (MPPEIP) architecture that supports all the angular prediction modes and prediction unit (PU) sizes is designed and tested on FPGA.

The rest of the chapter is organized as follows. Section 3.2, describes the design and implementation details of DC and planar intra predictions. In Section 3.2.3, we discuss the two architectures for DC and planar predictions, and their performance analysis is presented in Section 3.2.6. Section 3.3 covers the implementation details of the MPPEIP engine, and Section 3.3.3 presents the hardware architecture of directional intra predictions in detail. Experimental results of the MPPEIP engine and analysis is presented in Section 3.3.4. Section 3.4 concludes the chapter.

3.2 Implementation of planar and DC directional predictions

Intra prediction predicts the current PU using the neighbouring samples that are already predicted, capitalizing on the spatial redundancy present in the input video frame.

3.2.1 Planar prediction

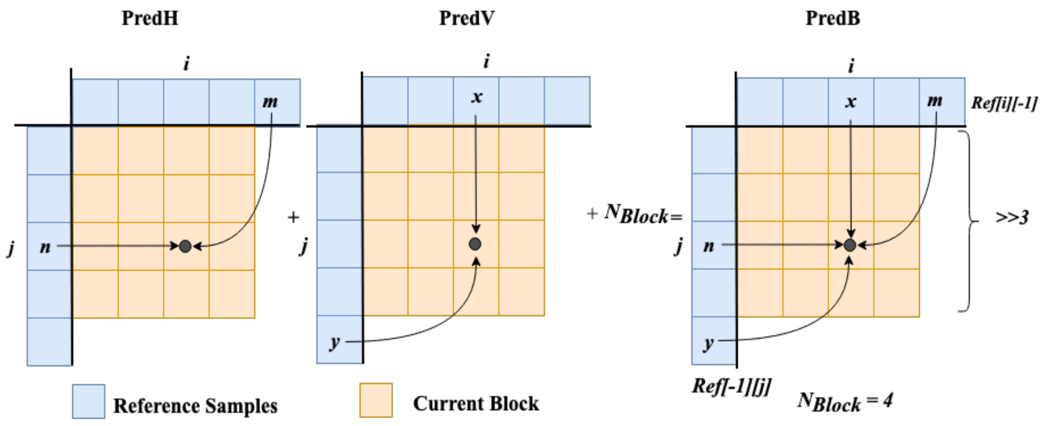


Figure 3.1: Planar prediction in a 4 × 4 PU

Planar mode predicts the samples by taking the weighted average of the horizontal and vertical reference samples (Wien (Jan. 2015), Sze *et al.* (2014)). This is illustrated in Figure 3.1 for a sample in location (i, j). This sample is the average of the four reference samples - m, n in the horizontal direction, and x, y in the vertical direction. Planar mode generates smooth prediction samples. The predicted sample, $PredB[i][j]$ is given by (3.1), where $PredH[i][j]$ and $PredV[i][j]$ are calculated using relations in

(3.2) and (3.3) respectively, where N_{Block} is the PU size, and $N_{Block} = 4, 8, 16, 32$. Planar prediction is the most expensive mode in terms of computations (Sze *et al.* (2014)).

$$PredB[i][j] = (PredH[i][j] + PredV[i][j] + N_{Block}) \gg (\log_2 N_{Block} + 1) \tag{3.1}$$

$$PredH[i][j] = (N_{Block} - 1 - i) \times Ref[-1][j] + (i + 1) \times Ref[N_{Block}][-1] \tag{3.2}$$

$$PredV[i][j] = (N_{Block} - 1 - j) \times Ref[i][-1] + (j + 1) \times Ref[-1][N_{Block}] \tag{3.3}$$

$$where,\ i, j = 0, 1, 2, ..., (N_{Block} - 1).$$

3.2.2 DC prediction

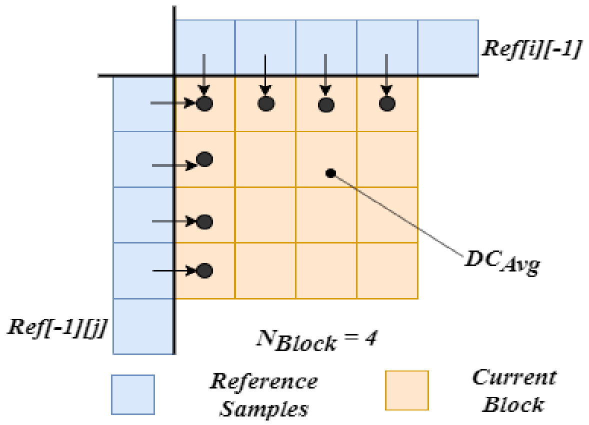


Figure 3.2: DC prediction in a 4×4 PU

In DC mode prediction, the average value of the top and left reference samples is generated as shown in Figure 3.2. The average value, DC_{avg}, is computed using the relation (3.4).

$$DC_{Avg} = \left(\sum_{i=0}^{N_{Block}-1} Ref[i][-1] + \sum_{j=0}^{N_{Block}-1} Ref[-1][j] \right) \gg (\log_2 N_{Block} + 1) \tag{3.4}$$

DC mode is applied to the flat regions of the video data. Filtering is applied to remove blocking effects on PU blocks smaller than the size 32×32. For PUs of size 32×32, all the samples in the current PU are replaced by DC_{avg}. In the case of 4×4, 8×8, 16×16 PUs, filtering is applied to corner, top-row, and the left-column reference samples using (3.5), (3.6) and (3.7), respectively. The corner sample smoothing is given by relation (3.5). Boundary samples of the top-row and the left-column are filtered using

(3.6) and (3.7) relations respectively. DC prediction is the least expensive mode in terms of computations.

$$PredB[0][0] = (PredB[-1][0] + 2 * DC_{Avg} + \quad PredB[0][-1] + 2) \gg 2 \tag{3.5}$$

Filtering for the top horizantal and the left vertical boundary samples are given by (3.6) and (3.7) respectively.

$$PredB[i][0] = (PredB[i][-1] + 3 * DC_{Avg} + 2) \gg 2 \tag{3.6}$$

$$PredB[0][j] = (PredB[-1][y] + 3 * DC_{Avg} + 2) \gg 2 \tag{3.7}$$

$$where, \; i, j = 0, 1, 2, ..., (N_{Block} - 1).$$

3.2.3 Implementation of area efficient hardware architectures of planar and DC predictions

Two hardware architectures, parallel pipelined architecture (PPA), and parallel datapaths architecture (PDA), are proposed, and implemented on FPGA and they support all the PU sizes. Planar and DC intra prediction operations can be grouped into three stages, 1) Reference samples selection, 2) Prediction and filtering of the boundary samples, and 3) Reconstruction. Reading the reference samples on to the FPGA board, selecting and storing them in buffers for prediction are the key challenges faced during hardware implementation.

3.2.4 Parallel Pipelined Architecture (PPA) for planar and DC predictions

PPA is designed using pipeline and parallel techniques and is shown in Figure 3.3. Reference samples reading, storing them into reference buffer procedures are pipelined. Planar and DC mode predictions are on the last stage of the pipeline and produce eight samples in parallel. One of the prediction modes is active at any given time. Reconstruction is pipelined, and a throughput of 8 samples in every clock cycle is produced.

In the *reference selection* module, corner, top, top-right, left and left-bottom reference

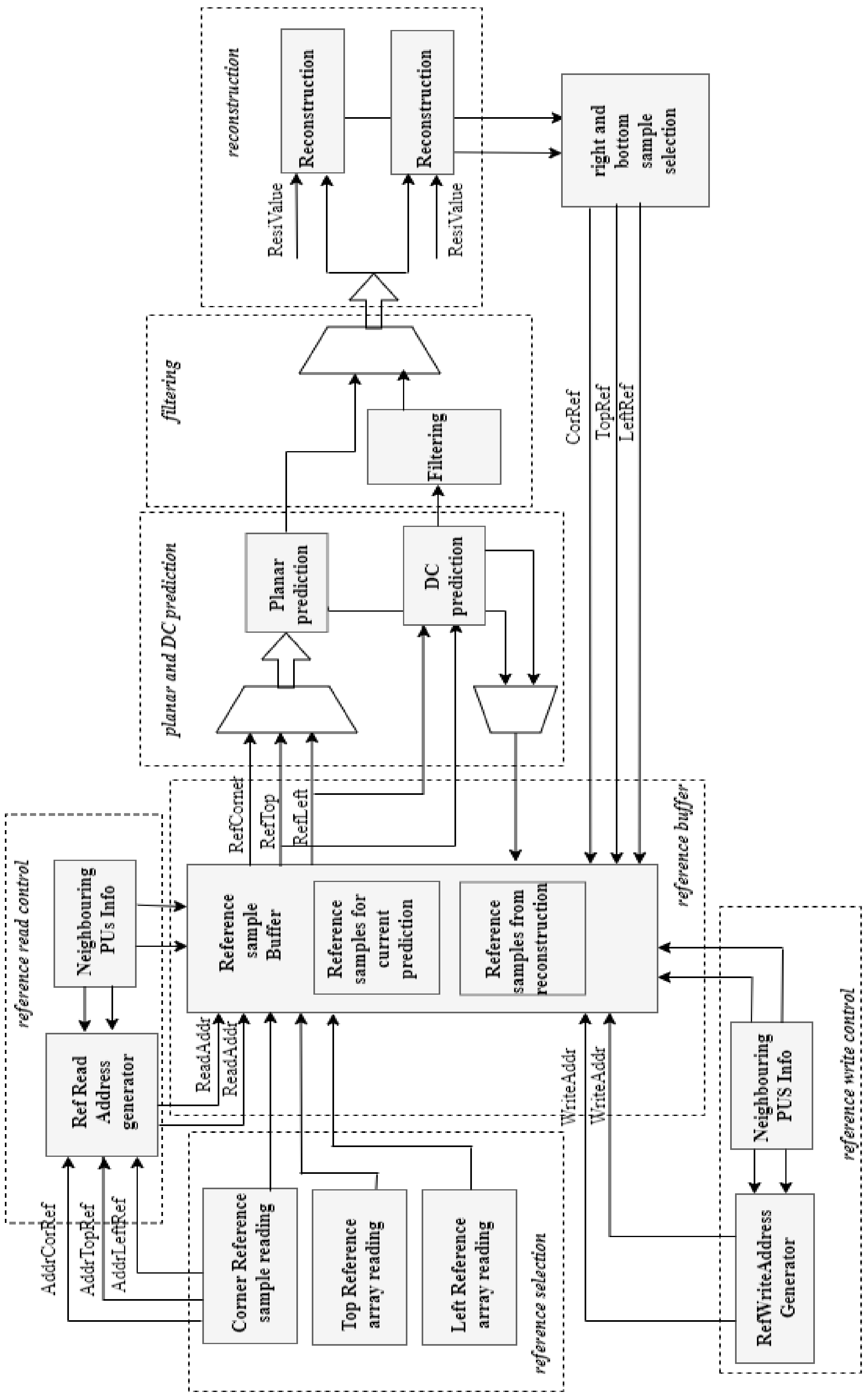


Figure 3.3: The proposed parallel pipelined architecture for planar and DC intra prediction

Table 3.1: Pipeline analysis of the PPA architecture for a 4×4 PU

Clock	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Reference selection	-	-	R1	R2	-	R1	R2	R1	R2	R1	R2	R1	R2	R1	R2	R1
Planar Prediction	-	-	-	-	PA1	PA2	PB1	PB2	PC1	PC2	PC1	PD1	PD2	P2	P2	P2
DC Prediction	-	-	-	-	PA1	PA2	PB1	PB2	PC1	PC2	PC1	PD1	PD2	P2	P2	P2
Filtering	-	-	-	-	-	PA1	PA2	PB1	PB2	PC1	PC2	PD1	PD2	P2	P2	P2
Reconstruction	-	-	-	-	-	-	RcA1	RcA2	RcB1	RcB2	RcC1	RcC2	RcD1	RcD2	Rc2	Rc2

$(R1, R2)$ - Reference samples P - Predicted Samples Rc - Reconstructed Samples

samples are selected and then stored in the *reference buffers*. The reference sample addresses are generated in the *reference write control* module. Samples are read and tracked using *reference read control* module. The information about neighbouring PUs, which is useful for the following predictions are stored in the *reference read control* module. Reference samples undergo prediction in the *planar and DC prediction* unit. Either planar or DC prediction is performed depending on the mode selected. The boundary samples in DC mode are filtered in the *filering* module. The predicted samples move to the next unit for reconstruction. Depending on lossy or lossless encoding, *ResiValue* is used for reconstruction. The boundary samples are written back into the reference buffers for future predictions.

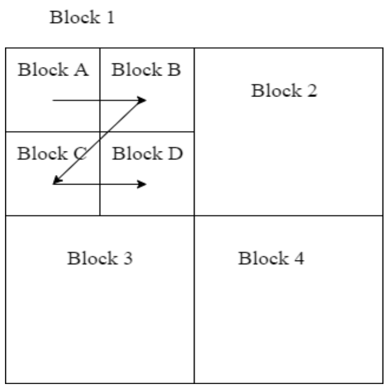


Figure 3.4: Prediction and scanning of PUs within a LCU

The pipeline analysis of the PPA engine for a 4×4 PU is shown in Table 3.1 and 8 samples are processed in each clock cycle. One corner sample, eight top, and eight left samples are required to predict one 4×4 PU. $R1$ and $R2$ indicate fetching of top, left and corner reference samples. PUs within a LCU are scanned in the Z order as shown in Figure 3.4, where, $BlockA, B, C, D$ are 4×4 PUs. Reading and writing samples into the reference buffers takes two lock cycles. In clock cycles, 3 and 4, reference samples are available in the buffers. In clock cycle 5, prediction starts. Based on the block position, the bottom row or the right column is predicted first, and the samples are sent to reconstruction and written back into reference buffers (to use for later predictions). Prediction is represented using P, $PA1 - PA2$ are predicted samples of PU block A. It takes 2 clock cycles to prediction one 4×4 PU. At clock cycle 6, filtering for DC mode takes place, the planar mode takes two clock cycles to complete prediction. In clock cycle 7, reconstruction of samples is performed which is represented using Rc. Boundary samples are written back into reference buffers, for future predictions. The transition to a larger PU size is seamless in the PPA engine, with no data delays and the produces the same throughput.

3.2.5 Parallel Datapaths Architecture (PDA) for planar and DC predictions

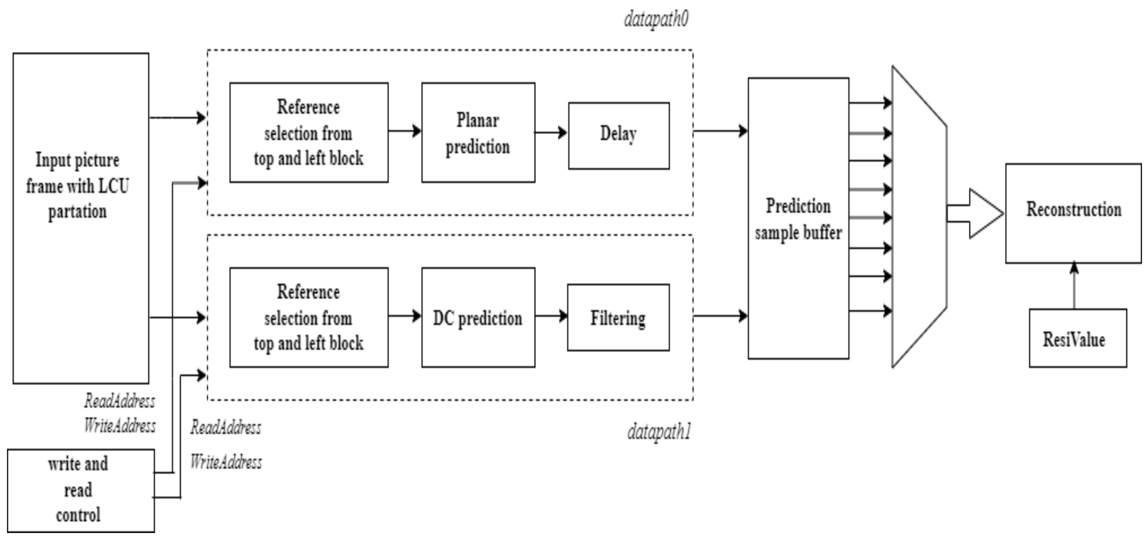


Figure 3.5: Proposed parallel datapath architecture for intra prediction

PDA engine consists of two parallel datapaths, as shown in Figure 3.5. *datapath0* and *datapath1* performs planar and DC predictions respectively. Both are independent units. At any given time, both datapaths or any one of the datapaths can be active depending on the prediction mode selected. Each datapath can process a throughput of 8 samples per clock cycle. Each datapath has *reference selection* unit, *reference* *buffer* and *angular prediction* unit. The predicted samples are stored in the *predicted sample buffer*, and reconstruction takes place. *write and read control* unit generates and stores the address of the reference samples.

Limitations of PPA and PDA architectures: PPA and PDA are designed for DC and planar directional modes. However, to extend these designs to support other directional modes, the architectures need several modifications. Reference samples are stored in on-chip memory, reading and writing the reference samples into these buffers causes latency.

3.2.6 Experimental results and analysis of PPA and PDA hardware architectures

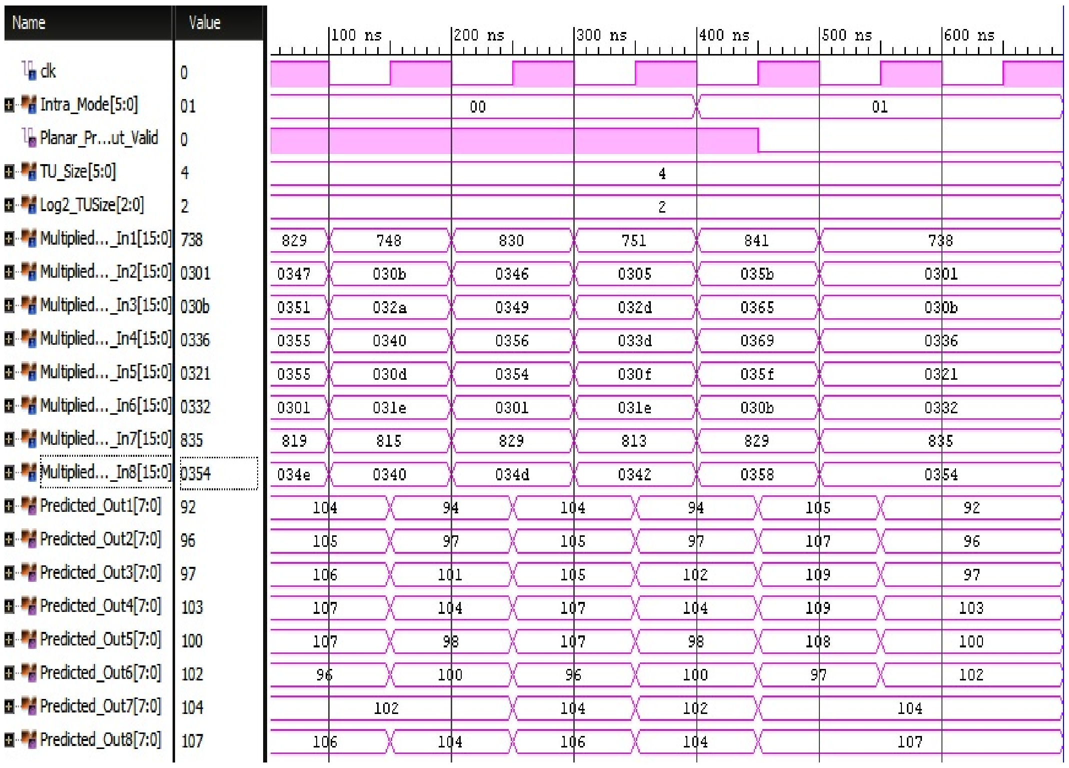
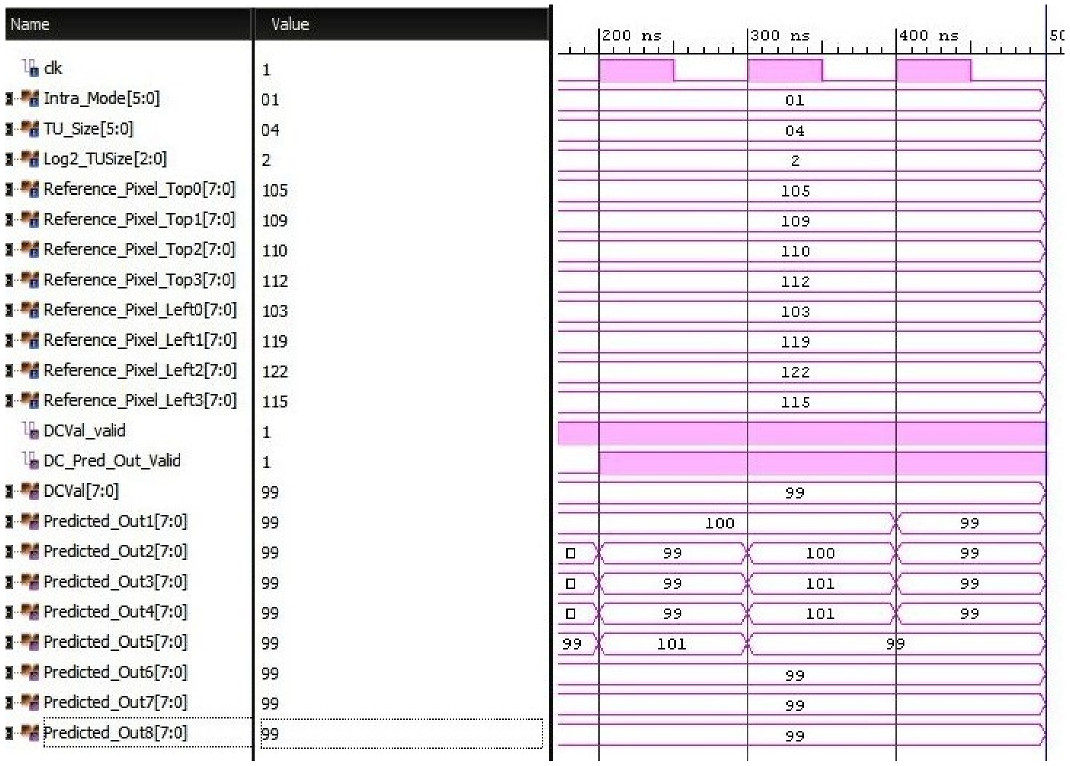


Figure 3.6: Simulation results of planar prediction using PPA for a 4 × 4 PU

Post-implementation reports for PPA and PDA are generated using Xilinx Vivado and tabulated in Table 3.2. The behavioural simulation results of planar and DC predictions using PPA architecture are shown in Figure 3.6 and Figure 3.7 respectively. The pipeline analysis and simulation results show that, the PPA and PDA



Name	Value
clk	1
Intra_Mode[5:0]	01
TU_Size[5:0]	04
Log2_TUSize[2:0]	2
Reference_Pixel_Top0[7:0]	105
Reference_Pixel_Top1[7:0]	109
Reference_Pixel_Top2[7:0]	110
Reference_Pixel_Top3[7:0]	112
Reference_Pixel_Left0[7:0]	103
Reference_Pixel_Left1[7:0]	119
Reference_Pixel_Left2[7:0]	122
Reference_Pixel_Left3[7:0]	115
DCVal_valid	1
DC_Pred_Out_Valid	1
DCVal[7:0]	99
Predicted_Out1[7:0]	99
Predicted_Out2[7:0]	99
Predicted_Out3[7:0]	99
Predicted_Out4[7:0]	99
Predicted_Out5[7:0]	99
Predicted_Out6[7:0]	99
Predicted_Out7[7:0]	99
Predicted_Out8[7:0]	99

Figure 3.7: Simulation results of DC prediction using PPA for a 4 × 4 PU

architectures in each clock cycle process and produce eight samples. In Table 3.2, the results for PUs of size 4, 8, 16 and 32 are tabulated. PU sizes are compared in terms of number of LUTs and registers used in the hardware designs, frequency of operation and throughput achieved.

Resource utilization: From Table 3.2, it can be noticed that PPA fares better than PDA in terms of the number of look-up tables (LUTs) and registers utilized by the designs. There is a decrease in LUT consumption of 20% when PDA is adopted instead of PPA configuration for 4 × 4 PUs. While PPA fares better for all other PU sizes. PPA uses 20%, 46% and 62% less resources for PUs of size 8, 16 and 32 respectively. It can also be noticed that PPA uses less registers than PDA.

Timing analysis: Pipeline analysis is shown in Table 3.1. Both in PPA and PDA designs, their is an initial delay for selecting and reading the reference samples. Once the prediction starts, samples are fed back to buffers, and prediction continues without any delays. Table 3.1 shows the pipelining analysis for 4 × 4 PU size. There is an initial delay of 2 cycles for PUs of size 4. Similarly, there is an initial delay of 4, 8,

Table 3.2: Comparison of the PDA and PPA architectures

	PDA	**PPA**
PU size = 4×4		
Number of LUTs	6.2K	8.2K
Registers	5.2K	4.2K
Frequency(MHz)	50.505	150
Throughput	8 pixels/cycle	8 pixels/cycle
PU size = 8×8		
Number of LUTs	10.3K	8.2K
Registers	8.7K	4.2K
Frequency(MHz)	51.813	150
Throughput	8 pixels/cycle	8 samples/cycle
PU size = 16×16		
Number of LUTs	15.3K	8.2K
Registers	13.1K	4.2K
Frequency(MHz)	43.478	150
Throughput	8 pixels/cycle	8 samples/cycle
PU size = 32×32		
Number of LUTs	21.7K	8.2K
Registers	15.4K	4.2K
Frequency(MHz)	41.478	150
Throughput	8 pixels/cycle	8 samples/cycle

and 16 cycles for PUs of sizes 8, 16, and 32, respectively.

The proposed hardware architectures supports all PU sizes while Abramowski and Pastuszak (2014) supports only 4×4 PUs and Li *et al.* (2011) shows implementation for only PUs of sizes 8 and 4. Min *et al.* (2017) is a fully pipelined architecture and supports all block sizes with a throughput of 4 pixels while the proposed design results in 8 samples in every clock cycle. Both PDA and PPA architectures produce a throughput of 8 samples per clock cycle. So, in applications where the area is a constraint, PPA is a better choice. If the video frames have large number of 4×4 PUs, then PDA is a better option. The same architectures can be extended to perform all the other intra angular predictions.

3.3 Design and implementation of the mixed parallel-pipelined efficient architecture of intra prediction (MPPEIP) engine for HEVC encoder

The proposed MPPEIP architecture supports all the 35 angular intra predictions. Intra angular predictions are much denser in horizontal and vertical directions than the diagonal directions (Sze *et al.* (2014)), (Wien (Jan. 2015)). This is an advantage as the horizontal and vertical directions occur more frequently in any picture frame than in the diagonal directions. A novel intra prediction engine is designed and implemented on FPGA with a combination of parallel and pipelining techniques. A high throughput of eight samples per clock cycle is engineered using the following techniques.

1. Efficient, reusable buffers are designed and managed for storing reference samples. The same buffers are reused by updating them with new values in the place of old ones and unused samples are discarded. This buffer structure supports all PU sizes and all angular prediction modes.

2. High throughput of 8 samples for each clock cycle is achieved to support 4 K video applications. The proposed design predicts eight samples in parallel and using pipeline techniques. The data loading delays are eliminated in the design.

3. Multiplication and addition operations in the prediction algorithm is carried out in a separate dedicated module to reduce the number of digital signal processing (DSP) slices used on the FPGA. The prediction algorithm structure has complex and parallel computations. This unit takes advantage of the parallelism to decrease hardware resources.

3.3.1 Semantics of directional intra predictions

In intra predictions, the basic unit used for prediction is the PU. The current PU is predicted using the coded neighbouring boundary samples which are called as reference samples. The steps outlined below are used to make angular intra predictions.

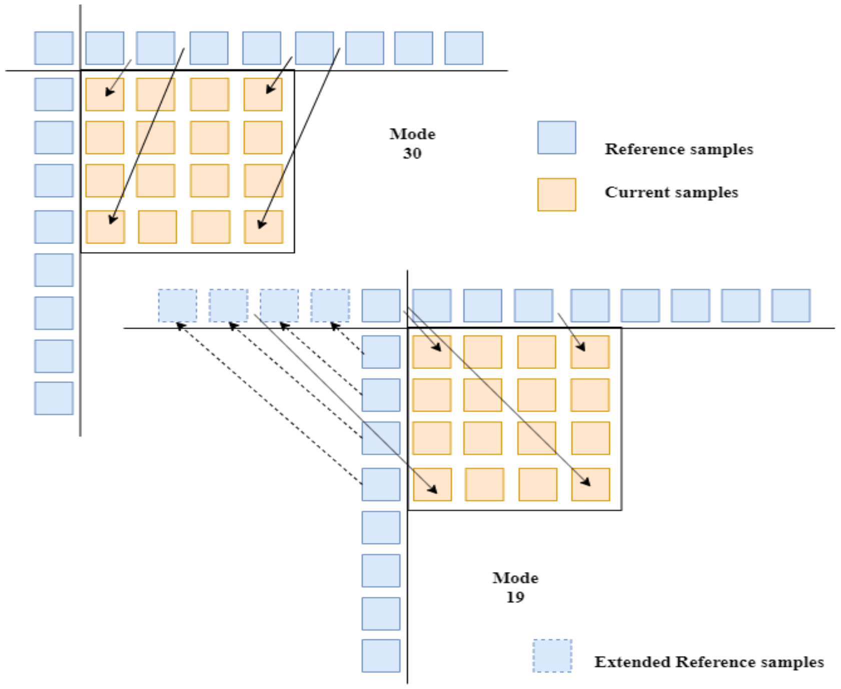


Figure 3.8: Illustration of reference samples selection. Top: Selection of references for positive vertical prediction mode 30. Bottom: Array extension of negative vertical prediction mode 19 and reference selection

3.3.1.1 Reference samples selection

Reference samples selection is performed as explained in Section 2.3.1 and reference array is generated. Reference sample index for vertical and horizontal angular predictions is generated using relations from (3.8) to (3.11). Here *ind* is the index of the reference samples used for prediction of a sample, and *fracPrt* is the weight of the reference samples. *Ang* in relations from (3.8) to (3.11) is the direction associated with each prediction. Intra prediction modes and the corresponding angles are tabulated in detail in Table 3.3.

$$ind = ((i+1) \times Ang) \gg 5, \text{ for horizontal directional modes} \tag{3.8}$$

$$ind = ((j+1) \times Ang) \gg 5, \text{ for vertical directional modes} \tag{3.9}$$

$$fracPrt = ((i+1) \times Ang) \,\&\, 31, \text{ for horizontal directional modes} \tag{3.10}$$

$$fracPrt = ((j+1) \times Ang) \,\&\, 31, \text{ for vertical directional modes} \tag{3.11}$$

Table 3.3: Intra prediction modes, their associated directions and inverse angles

Mode (Horizontal)	**Ang**	Inverse Angle	**Mode** (Vertical)	**Ang**	Inverse Angle
2	32		18	-32	-256
3	26		19	-26	-315
4	21		20	-21	-390
5	17		21	-17	-482
6	13		22	-13	-630
7	9		23	-9	-910
8	5		24	-5	-1638
9	2		25	-2	-4096
10	0		26	0	
11	-2	-4096	27	2	
12	-5	-1638	28	5	
13	-9	-910	29	9	
14	-13	-630	30	13	
15	-17	-482	31	17	
16	-21	-390	32	21	
17	-26	-315	33	26	
			34	32	

where $\gg$ is the bitwise shift right, and & is the bitwise AND operations.

Reference samples are selected from the reference array ($RefAng$). Every angular prediction requires a unique reference array, which must be computed for each mode. Reference array is classified as the main array and the extended main array. The main array for horizontal predictions is the reference samples on the left side of the current PU. Positive horizontal predictions (modes 2 to 10) use only the main array (left side samples). Negative horizontal predictions (Modes 11-17) use an extended main array. The main array is expanded to accommodate the samples projected from the top array to form an extended main array. Similarly, for vertical predictions, the samples above the current PU are used to form the main array, and the samples from the left are projected on to the top, to form the extended main array as illustrated in Figure 3.8. The negative vertical predictions (modes 18-25) use extended main array, and the positive vertical predictions (modes 26-34), use the main array. Relations (3.12) and (3.13) are used to build the main array for the vertical and horizontal modes, respectively. While the extended main array is generated using the relations (3.14) and (3.15).

$$RefAng[i] = Ref[-1+i][-1], (i \geqslant 0) \text{ for vertical modes} \tag{3.12}$$

$$RefAng[j] = Ref[-1][-1+j], (j \geqslant 0) \text{ for horizontal modes} \tag{3.13}$$

$$RefAng[i] = Ref[-1][-1+((i \times IAng + 128) \gg 8)], (i < 0) \text{ for vertical modes} \tag{3.14}$$

$$RefAng[j] = Ref[-1+((j \times IAng + 128) \gg 8)][-1], (j < 0) \text{ for horizontal modes} \tag{3.15}$$

$IAng$ in (3.14) and (3.15) is inverse angle. Table 3.3 shows the mapping of these inverse angles to their corresponding modes.

3.3.1.2 Reference samples filtering

After the reference array is built, the samples in the array are filtered based on the PU size and the directional predictions to reduce the blocking effects. Filtering of reference samples is explained in Section 2.3.2.

3.3.1.3 Angular predictions

Planar and DC predictions are better suited for smooth and gradually changing video content, but they are not very useful for the high frequency and complex textured content (Wien (Jan. 2015)), (Lainema *et al.* (2012)). In such cases, angular predictions are most useful. The filtered reference samples are used for prediction. Planar and DC predictions are explained in Sections 3.2.1 and 3.2.2 respectively. There are 33 intra angular predictions in which the samples in a PU are predicted by interpolation of the reference samples using relations (2.10) and (2.11). Prediction of samples in the current PU is performed as explained in Section 2.3.3.

After prediction, the samples are reconstructed using predicted samples and residuals. The residue is the difference between the current and predicted samples. The reconstructed samples act as the reference data for later perditions.

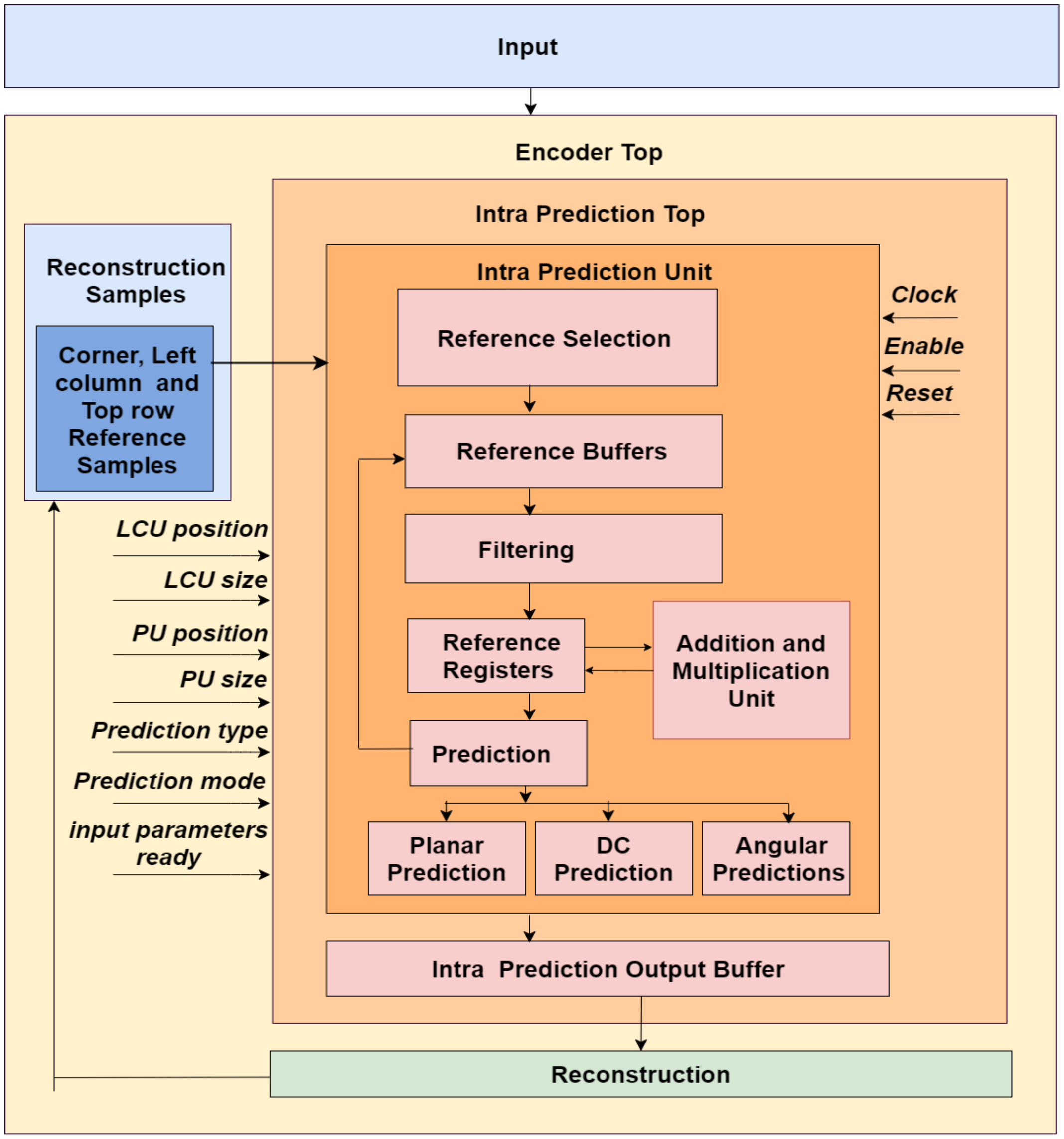


Figure 3.9: Block diagram of the top level architecture of the MPPEIP engine

3.3.2 Implementation of the proposed MPPEIP architecture

A mixed pipelined-parallel technique is introduced in the intra prediction unit that can achieve a throughput of eight samples in each clock cycle. The proposed MPPEIP architecture operates without any data dependency. The block diagram of the top level architecture of MPPEIP engine is shown in Figure 3.9. The input is fed to the *Encoder Top* layer. The *Encoder Top* layer generates and schedules the necessary control signals to coordinate all the modules present in the encoder. The input samples and signals for the intra prediction module are supplied from the *Encoder Top*. The inputs to the

MPPEIP engine are reference samples and details about the LCU position, LCU size, PU size, PU position, prediction type (horizontal or vertical) and prediction mode, which come from the top layer. Samples from the rightmost column and the bottom row are selected from the *Encoder top* to start the prediction of a PU. The *Intra Prediction Top* module generates clock, enable, and reset signals to control the *Intra Prediction Unit* and also communicates the control signals between the top layer and the underlying *Intra Prediction Unit*. The *Intra Prediction Unit* predicts the PUs and stores the predicted results in the *Intra output buffer block*. These samples are reproduced in the *Encoder Top*. Once the input reference samples and the input parameters necessary are available at the *Intra Prediction Unit*, the input parameters ready signal is made high and the prediction starts.

3.3.2.1 Reference Selection

This module selects the necessary reference samples for the prediction of current PU. The corner, top, top-right, bottom, and bottom-left samples are selected from the neighbouring PUs. A 5-bit register is used to indicate the availability of all these samples.

3.3.2.2 Reference Buffers

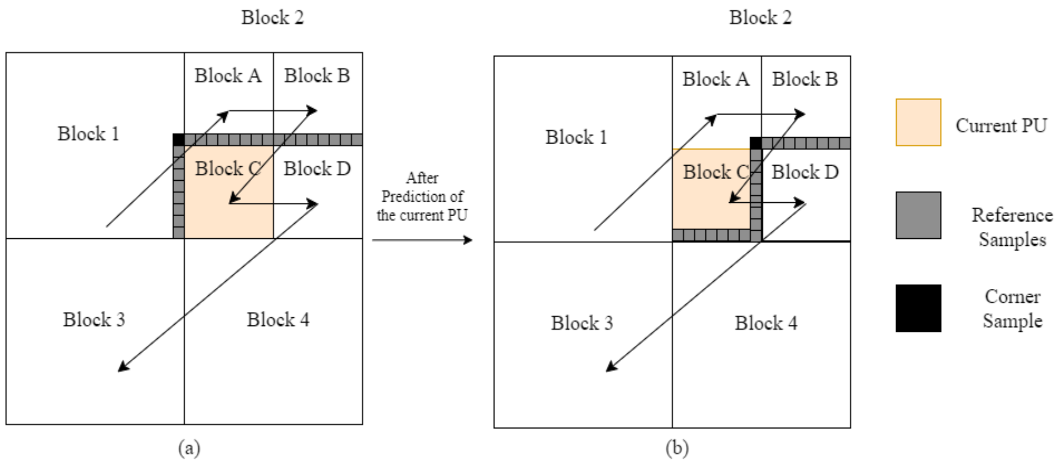


Figure 3.10: Illustration of updating reference buffers. (a) Reference buffer before the current PU is predicted. (b) Reference buffer after the current PU is predicted

The reference samples are filled into the reference buffers from neighbouring PUs and it is updated in a 5-bit register. The necessary buffer addresses are generated to facilitate the data exchange between the modules. Using buffer data, current PU is predicted. The border samples from this PU become the new reference data and they replace the older ones in the buffers. PUs are scanned in the Z-order as shown in the Figure 3.10. The current PU that is under prediction and the reference samples of interest are highlighted. Block C is the PU under prediction and it is located within Block 2. Prediction is already completed in Block 1, Block A and Block B. Hence the corner (C) sample, top (T), top-right (TR), left (L) samples are available in the reference buffers, as shown in Figure 3.10 (a). Left-bottom (LB) samples are unavailable and reference extension is performed. Using these samples, current PU is predicted. After the prediction, the border samples from Block C replaces the data in the reference buffer, as illustrated in Figure 3.10 (b). The same reference buffers are reused to store new data. Few samples get overlapped, few are replaced and unused older samples are discarded. In the Figure 3.10, sample C gets replaced, older TR becomes the new T samples, old L gets discarded and new samples take its place. The size of the reference buffer used in the design is limited to 1032 bits.

3.3.2.3 Filtering

The reference samples are filtered before the prediction. A set of condition checks are performed in this module to decide between strong filtering and weak filtering based on the PU size, as explained in Section 2.3.2. Once the filtering type is decided, filtering of the reference samples is carried out. This new set of filtered reference samples are moved to the next module for prediction.

3.3.2.4 Reference Registers

The function of this module is to exchange data with the *Addition and Multiplication Unit* based on the prediction modes. The filtered reference sample data is sent and the accumulated data is collected back.

3.3.2.5 Addition and multiplication unit

This module performs arithmetic operations necessary for intra prediction. Angular prediction computations are consistent for all the modes and for all PU sizes. For the prediction of each sample, the filtered reference samples are multiplied, accumulated and then shifted. In this module, filtered reference samples are selected based on the prediction mode, then multiplied and added. These accumulated values are sent to the *Reference registers* module. This module helps to significantly reduce the number of DSP slices used by the proposed MPPEIP design. A multiplexer logic is used to feed the input to the DSP slice and also to tell the DSP slice to perform addition or multiplication. To predict one sample, two multiplication and three addition operations are required, i.e to predict just one 4×4 PU, 32 multiplication and 48 addition operations are required. By using a dedicated module for these operations, the number of DSP slices used in the MPPEIP is reduced to a minimum.

3.3.2.6 Intra angular predictions

In this module, sample by sample prediction is performed. There are three prediction modules in parallel, Viz. Planar, DC, and Angular Predictions.

Planar Prediction: In this module, each sample involves multiplication followed by accumulation of the multiplied results as explained in Section 3.2.1. The module produces an output of 8 samples in each single operation cycle.

DC Prediction: This module executes two operations to process 8-samples in parallel.

1. Calculation of DC_{avg}: DC_{avg} value for the current PU is calculated using the Top and Left filtered reference samples using the relation (3.4).
2. Filtering of boundary samples and output calculation: Filtering conditions for the current PU block are checked. If the PU size is 32, then DC_{avg} is the final output, and all the samples in the current PU are replaced with DC_{avg}. In the case of chroma PUs, DC_{avg} is the final output. For other size luma PUs, the final output is the filtered samples of the first row and the first column of the current PU. The filtered output is generated using the relations (3.5) - (3.7) as explained in Section 3.2.2.

Angular Predictions: This module performs angular predictions in three steps processing eight samples in parallel.

1. Filling the reference array: The angular modes needs reference data in a unique way compared to planar and DC predictions. The reference samples array range

for every prediction mode is calculated. The maximum range of the array can extend from -64 to $+64$. Following the range calculation, reference samples array is selected and stored in registers. Reference samples array for all angular prediction modes and all the possible PU sizes are pre determined and stored in registers and they are used to fill up the register buffers as and when required. This is possible as angular modes and PU sizes are fixed and pre allocation of reference arrays reduces the number of cycles required to load the reference samples. Data loading delays are completely eliminated and the savings in terms of clock cycles is as follows. For one 4×4 PU, 8 Top reference samples, 8 right reference samples and, 1 corner sample is required for prediction. It requires 3 clock cycles to load the data operating at 8 samples per cycle. Similarly, for one 8×8 PU, 33 reference samples are required, and it takes 5 clock cycles. Similarly, in the case of one 16×16 PU, 65 reference samples needs to be fetched, and it requires 9 clock cycles. A 32×32 PU needs 129 reference samples, and it takes 17 clock cycles for data loading. These data loading delays are completely eliminated in the proposed MPPEIP design.

2. Calculating index and weight: Two reference samples at locations *ind* and $(ind+1)$ are required to predict one sample in the current PU. $fracPrt$ gives the weight for the references. The *ind* and $fracPrt$ values are required to pick the appropriate reference data from the buffers and to apply prediction based on the prediction mode.

3. Prediction and output calculation: The final predicted output is computed using the selected reference samples, *ind* value, $fracPrt$ value, angular mode and PU size. The border samples from the predicted PU are fed back to the reference buffers and used to predict the next PU.

3.3.2.7 Intra Prediction Output Buffer

Output samples from the intra prediction block are concatenated and stored in the output buffers after completing the prediction. The sample values are stored in buffers to be accessed by other modules of *Encoder Top* that use the predicted samples. This module generates the *output valid* signal to indicate the availability of output.

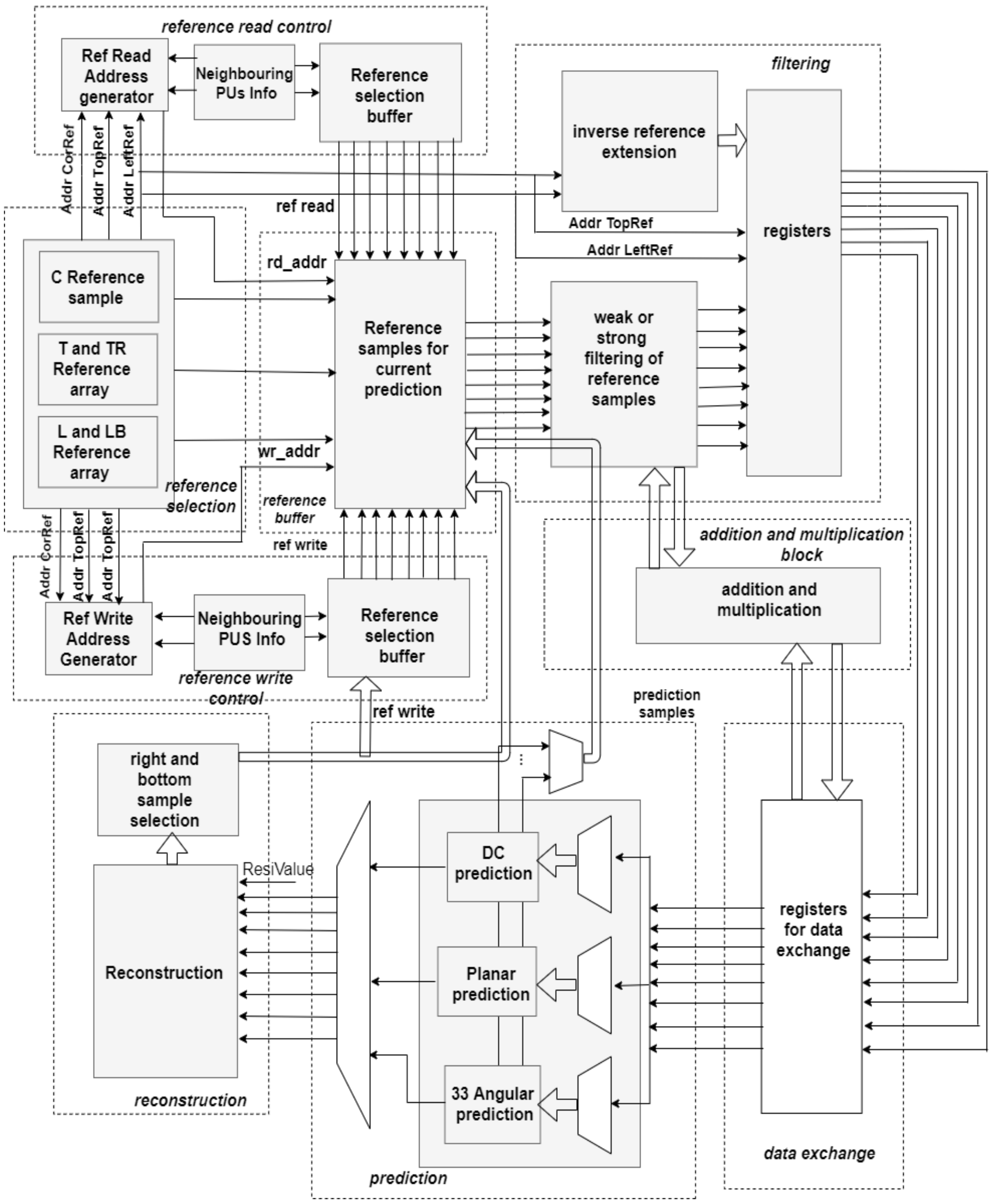


Figure 3.11: Block diagram of detailed architecture of the MPPEIP engine

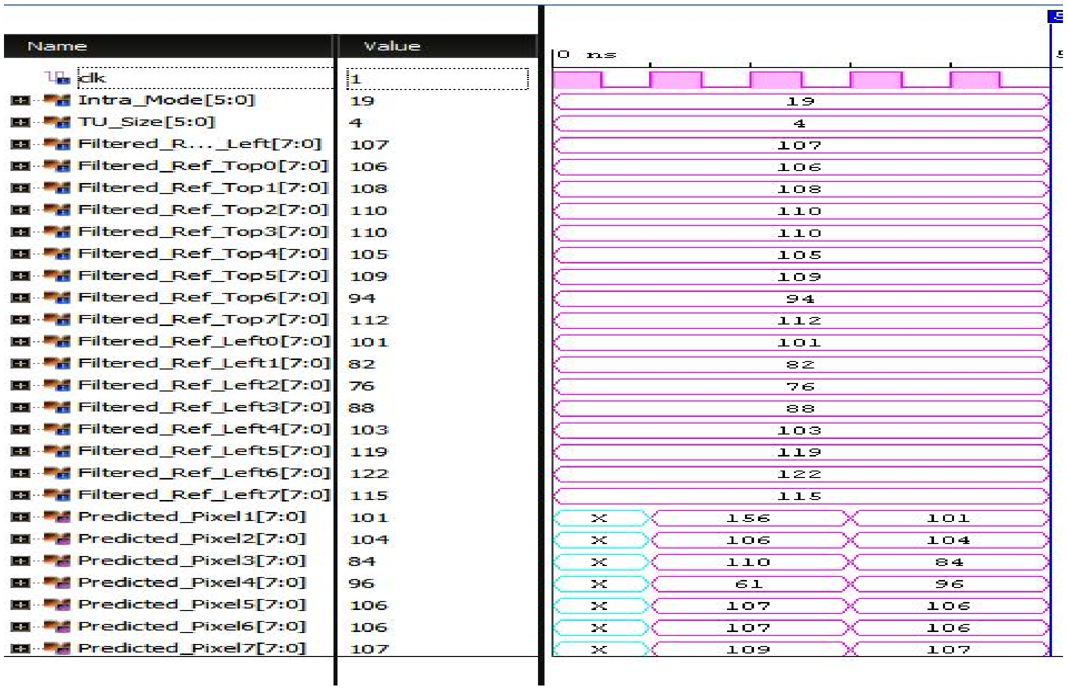
3.3.3 Detailed architecture of the proposed MPPEIP engine

The detailed architecture of the proposed MPPEIP engine is shown in Figure 3.11. The reference samples are read into the *reference selection* module. The *neighbouring*

PUs reg module contains the information of the neighbouring corner, top, top-right, left, left-bottom samples and also indicates the availability of these samples. The read and write addresses are generated for these samples. *Addr TopRef* is the address for both T and TR samples together and *Addr LeftRef* is the address for both L and LB samples. Once the selection of reference samples is complete, it is indicated by a 5-bit register. After selection, the samples are stored in the *reference buffer* module. The *reference read control* module generates the corresponding read addresses *rd_addr* for the buffer data. The *reference write control* module generates the necessary write addresses *wr_addr* for these samples.

Based on the PU size and the prediction mode, reference samples are moved into the *filtering* module. Decision to perform weak or strong filtering is made in this module. Accordingly, samples are filtered and stored in the registers. If smoothing is not required, the samples are directly read into the registers. For negative angular modes, the extended main array is computed, selected and stored in the *Extended Reference Array* module. The samples at *Addr, TopRef, Addr LeftRef* are used to generate extended main array for horizontal and vertical negative modes. The filtered reference samples are moved into the *reference registers* module. This module exchange data with the *addition and multiplication unit* module. In *addition and multiplication unit*, the arithmetic operations required for the prediction are carried out. The *reference registers* module sends filtered reference data and collects the accumulated data from the *addition and multiplication unit.* Arithmetic operations in the *addition and multiplication unit* are performed using DSP slices. The proposed design uses just forty DSP slices. The accumulated results in the *reference registers* module are passed on to perform prediction.

The current PU is predicted in the *prediction* module by either *Planar*, *DC*, or *Angular predictions*, depending on the prediction mode selected. Eight samples are predicted in parallel during every clock cycle. The bottom row and the right column samples of the current PU are used as the reference samples for the next PU prediction. They are written back to the *reference buffers.* The next PU starts the prediction without having to wait for the reconstruction of samples. The predicted samples are concatenated and stored in the *intra prediction output buffers.* It takes just 2 cycles to predict one 4×4 PU, 4 clock cycle for one 8×8 PU prediction, 32 cycles for one 16×16 PU and 128 cycles to predict one 32×32 size PU. The samples from the



```
Name                          Value   0 ns
clk                           1
Intra_Mode[5:0]               19      19
TU_Size[5:0]                  4       4
Filtered_R..._Left[7:0]       107     107
Filtered_Ref_Top0[7:0]        106     106
Filtered_Ref_Top1[7:0]        108     108
Filtered_Ref_Top2[7:0]        110     110
Filtered_Ref_Top3[7:0]        110     110
Filtered_Ref_Top4[7:0]        105     105
Filtered_Ref_Top5[7:0]        109     109
Filtered_Ref_Top6[7:0]        94      94
Filtered_Ref_Top7[7:0]        112     112
Filtered_Ref_Left0[7:0]       101     101
Filtered_Ref_Left1[7:0]       82      82
Filtered_Ref_Left2[7:0]       76      76
Filtered_Ref_Left3[7:0]       88      88
Filtered_Ref_Left4[7:0]       103     103
Filtered_Ref_Left5[7:0]       119     119
Filtered_Ref_Left6[7:0]       122     122
Filtered_Ref_Left7[7:0]       115     115
Predicted_Pixel1[7:0]         101     X   156   101
Predicted_Pixel2[7:0]         104     X   106   104
Predicted_Pixel3[7:0]         84      X   110   84
Predicted_Pixel4[7:0]         96      X   61    96
Predicted_Pixel5[7:0]         106     X   107   106
Predicted_Pixel6[7:0]         106     X   107   106
Predicted_Pixel7[7:0]         107     X   109   107
```

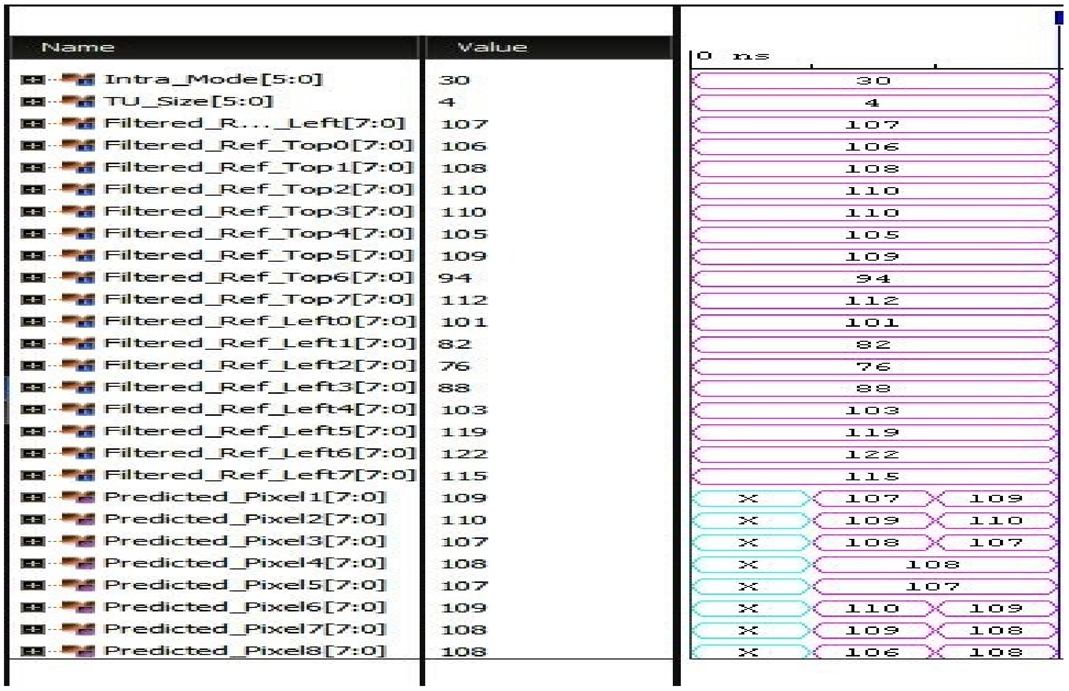
Figure 3.12: Mode 19 angular prediction simulation results

intra prediction output buffers module are sent to the *reconstruction* module. This module reconstructs the predicted samples using residue, *ResiVal*. The bottom row and right column samples of the PUs are written back into the *reference buffers* for later predictions. The entire architecture operates in a pipeline and there is no data dependency between any modules. Eight samples are produced in each clock cycle and there is no waiting period for data loading in any intermediate cycles.

Limitations of the MPPEIP architecture: The MPPEIP engine supports HD video applications. MPPEIP can be modified to be flexible, so that this engine can be used for a range of target applications but at the cost of using more hardware resources.

3.3.4 Experimental results and analysis of the MPPEIP hardware architecture

Behavioural simulation results of the intra angular predictions of modes 19 and 30 are shown in Figures 3.12 and 3.13 respectively. The simulation results show the eight samples predicted in every clock cycle. The proposed MPPEIP engine and other



```
Name                            Value    0 ns
Intra_Mode[5:0]                 30       30
TU_Size[5:0]                    4        4
Filtered_R...._Left[7:0]        107      107
Filtered_Ref_Top0[7:0]          106      106
Filtered_Ref_Top1[7:0]          108      108
Filtered_Ref_Top2[7:0]          110      110
Filtered_Ref_Top3[7:0]          110      110
Filtered_Ref_Top4[7:0]          105      105
Filtered_Ref_Top5[7:0]          109      109
Filtered_Ref_Top6[7:0]          94       94
Filtered_Ref_Top7[7:0]          112      112
Filtered_Ref_Left0[7:0]         101      101
Filtered_Ref_Left1[7:0]         82       82
Filtered_Ref_Left2[7:0]         76       76
Filtered_Ref_Left3[7:0]         88       88
Filtered_Ref_Left4[7:0]         103      103
Filtered_Ref_Left5[7:0]         119      119
Filtered_Ref_Left6[7:0]         122      122
Filtered_Ref_Left7[7:0]         115      115
Predicted_Pixel1[7:0]           109      X   107   109
Predicted_Pixel2[7:0]           110      X   109   110
Predicted_Pixel3[7:0]           107      X   108   107
Predicted_Pixel4[7:0]           108      X      108
Predicted_Pixel5[7:0]           107      X      107
Predicted_Pixel6[7:0]           109      X   110   109
Predicted_Pixel7[7:0]           108      X   109   108
Predicted_Pixel8[7:0]           108      X   106   108
```

Figure 3.13: Mode 30 angular prediction simulation results

state-of-the-art counterparts are compared in terms of supported PU sizes, technology used, clock speed, throughput, hardware cost (in terms of LUTs and registers used by the design), supported video applications, and the reference buffer size used in the designs in Table 3.4. In Table 3.4 works (Huang *et al.* (2014)), (Chiang *et al.* (2016)), (Pastuszak and Abramowski (2016)), (Fan *et al.* (2019)) represent implementations using Application-specific Integrated Circuits (ASIC) and works (Amish and Bourennane (2016)), (Choudhury and Rangababu (2017)), (Min *et al.* (2017)) are all implementations on FPGA. In (Huang *et al.* (2014)), the architecture produces a throughput of four samples. But reference data loading requires additional clock cycles, limiting the throughput to 2.46 samples/cycle. Due to the implementation on ASIC platform, frequency is much higher. In (Chiang *et al.* (2016)), the entire HEVC decoder is designed and implemented on ASIC platform. Table 3.4 lists only intra unit data. The throughput is limited to 1.78 samples/cycle due to data dependency. The frequency is higher due to the ASIC implementation, but the design area, the buffer size is much larger than the proposed design. In (Pastuszak and Abramowski (2016)), the design uses 64 processing units in one intra encoder datapath and 16 processing units in another datapath for prediction. However, the throughput is lim-

Table 3.4: Comparison of the proposed MPPEIP engine with other similar works

Parameters	Huang *et al.* (2014)	Chiang *et al.* (2016)	Pastuszak and Abramowski (2016)	Fan *et al.* (2019)	Amish and Bourennane (2016)	Choudhury and Rangababu (2017)	Min *et al.* (2017)	Proposed MPPEIP engine
Implementation	ASIC	ASIC	ASIC	ASIC	FPGA	FPGA	FPGA	FPGA
Block size	all	all	all	all	all	all	all	all
Technology	40 nm	90 nm	90 nm	65 nm	28 nm	28 nm	65 nm	28 nm
Max Freq (MHz)	200	270	200	400	219	119	219	110
Throughput samples/cycle	2.46	1.78	16	32	0.2	16	4	**8**
Hardware Cost (ASIC-gates/FPGA-LUTs)	27 K	77 K	127.3 K	66.2 K	170 K	112 K	14 K	**16 K**
slice registers	-	-	-	-	110 K	12 K	5.5 K	122
Video spec	4 K	4 K	4 K	8 K	4 K	4 K	4 K	4 K
frames per second	30	30	30	30	24	30	30	30
Buffer size	4.9 KB	5.6 KB	6.0 KB	0.8 KB	–	–	6.0 KB	**1.0 KB**

ited to 16 samples at the cost of more hardware and a larger buffer memory size of 6 KB. In (Fan *et al.* (2019)), the architecture has 32 parallel processors in parallel for prediction. More units in parallel results in requirement of more hardware resources. However, with the 32 samples throughput, 2 clock cycles are utilised to process one 4×4, which is same as our proposed design. In (Amish and Bourennane (2016)), five parallel paths are implemented to achieve high throughput. However, the strategy is limited to 4 K @ 24 fps video applications. Also, the design uses 77 K LUTs and 110 K registers. This design does not use register buffers and is implemented on Virtex-6 and 7 FPGA platforms. In (Choudhury and Rangababu (2017)), throughput is 16 samples/cycle. Due to data loading, it takes 6 cycles to predict a 4×4 block making the throughput average to 2.33 samples/cycle. Also, the parallel structure uses 112 K LUTs and 112 K registers. In (Min *et al.* (2017)), the number of LUTs used is slightly less than our proposed design, but considerably more slice registers are used and the reference buffer memory size is 6 KB with just 4 samples/cycle throughput.

3.4 Summary

The proposed planar and DC architectures produce a throughput of 8 samples in every operation cycle which can process and support HD video applications. Based on the simulation analysis of PDA and PPA architectures the conclusions drawn are, in applications where the area is a constraint, PPA is a better choice. If the video frames have large number of 4×4 blocks, then PDA is a better option. Since the PPA architecture produce better throughput using optimum silicon area irrespective of PU size, this technique was extended to support all the angular intra predictions.

The proposed MPPEIP engine is designed to produce eight samples per clock cycle in parallel and operates in a full pipeline. A compact reusable reference buffer of size 1 KB is implemented that reduces the buffer memory size. Data loading of reference samples causes significant delays limiting the throughput, which is eliminated by assigning the reference data manually for all the angular modes and all PU blocks as they are fixed. A dedicated unit for arithmetic operations is introduced to reduce the number of DSP slices used and to optimise the design. The proposed MPPEIP design uses only 40 DSP slices. These two optimisations in the proposed design play a key role to significantly reduce the resources used on the hardware. The hardware resources used by the proposed MPPEIP design is 16 K LUTs and 122 registers, which is considerably less than most of the existing works.

Chapter 4

Implementation of an efficient parallel-pipelined intra prediction architecture to support DCT/DST engine of HEVC encoder

Everything should be made as simple as possible, but not simpler.

— Albert Einstein, Theoretical physicist

4.1 Introduction

Intra prediction and discrete cosine transform/discrete sine transform (DCT/DST) modules are two fundamental and consecutive units that play a vital role in improving the compression performance of the HEVC encoder. According to the encoding standard, the input/output formats of the intra module and DCT/DST engines are different. Intra module process the PU blocks both row-wise and column-wise based on the directional modes. The DCT/DST module process row-wise samples from the variable sized TUs. Due to this inconformity in the data exchange between both the modules, the MPPEIP engine cannot be utilised to its full potential. In this chapter a novel area-efficient parallel-pipelined intra prediction architecture to support DCT/ DST engine is proposed.

Due to the inconformity in data processing of intra prediction and DCT/DST engine the output from the intra prediction needs to be stored in an memory buffer. The intermediate buffer is shown using dotted lines in Figure 4.1. Generally, in horizontal predictions, column-wise samples are produced which cannot be directly processed by the DCT/DST engine and needs to wait until the entire block is predicted. This affects the overall throughput of the HEVC encoder, which is not utilising the advantages of intra module fully. Apart from this, using the intermediate buffer is an unnecessary hardware overhead. Hence in this chapter an intra prediction design which always process row-wise samples so that they can be directly transform coded is proposed.

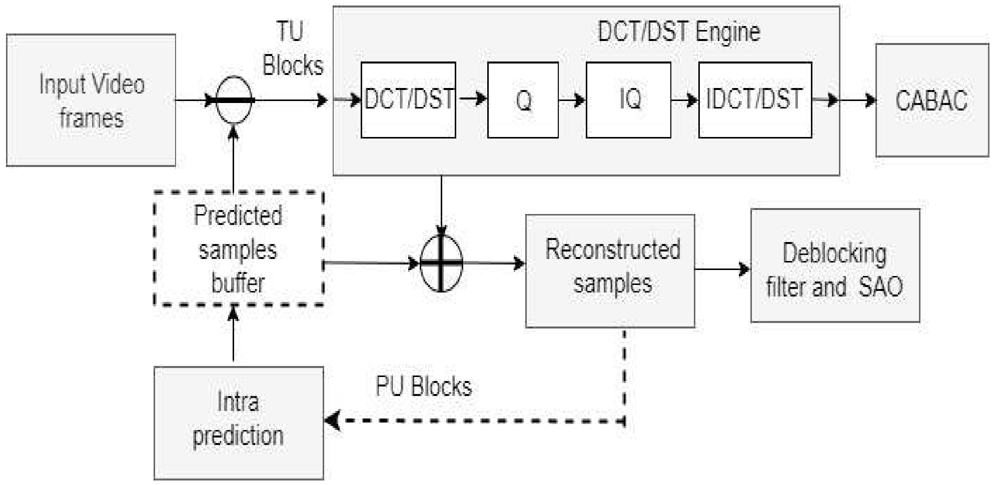


Figure 4.1: Block diagram of HEVC encoder with the predicted samples buffer shown using the dotted lines

Analysing the implementations available in the literature and keeping the current HD/UHD applications in view, the following observations are drawn.

- It is important to design an intra prediction engine that can support all the PU sizes and all the directional modes while achieving high throughput.
- Adding more parallel processing elements may increase the throughput but that comes with an increased hardware cost and power requirements. Hence it becomes very crucial to maintain a balance among the parallel/pipeline/sequential elements used in the design to achieve an efficient hardware engine.
- Most of the hardware architectures implemented in the literature requires an

intermediate reference buffer between the intra prediction and the DCT/DST engines. This is because the intra prediction output is in the form of PU blocks whereas DCT/DST engine requires TU blocks as inputs. Furthermore, the samples to the DCT/DST engine is scanned row-wise first and then column-wise from variable sized TUs.

In order to cater to all the above requirements, an efficient improved intra prediction architecture is proposed with the following key techniques, which are the major contributions of this chapter.

1. The intra prediction engine is configured to generate 8 samples in parallel in every clock cycle with support to all 35 directional modes and all possible PU sizes. Further, this novel intra prediction engine is implemented on the FPGA to achieve a high throughput to support 4 K videos. A reconfigurable compact reference buffer to hold the source references is included in the engine, which reduces the time required to fetch the references considerably.
2. A fully pipelined DCT/DST based intra prediction engine is proposed. The samples are always operated and generated row-wise in the intra prediction module. With this, DCT/DST and intra prediction engines can operate in parallel ensuring the high throughput.
3. The hardware consuming interstage memory buffer between transform and intra prediction modules is eliminated by introducing row-wise processing of samples in the proposed design.
4. Arithmetic operations in the directional angular predictions are one of the causes to increase the complexity of intra prediction engine. The prediction function is uniform for all 33 directional modes. Hence, we have proposed a dedicated module to process multiplication and addition operations to ensure the reuse of multipliers present in the digital signal processing (DSP) slices of the FPGA, which in turn improves the efficiency of the hardware engine.

The rest of the chapter is organized as follows. The semantics of intra prediction encoding procedure is discussed in Section 4.2. Next, the proposed hardware architecture of the DCT/DST based intra prediction engine and its timing analysis are explained in Section 4.3. The detailed performance analysis of the proposed DCT/DST based prediction engine is presented and discussed in Section 4.4. Finally, conclusions are drawn in Section 4.5.

4.2 Semantics of the DCT/DST based intra prediction engine in HEVC encoder

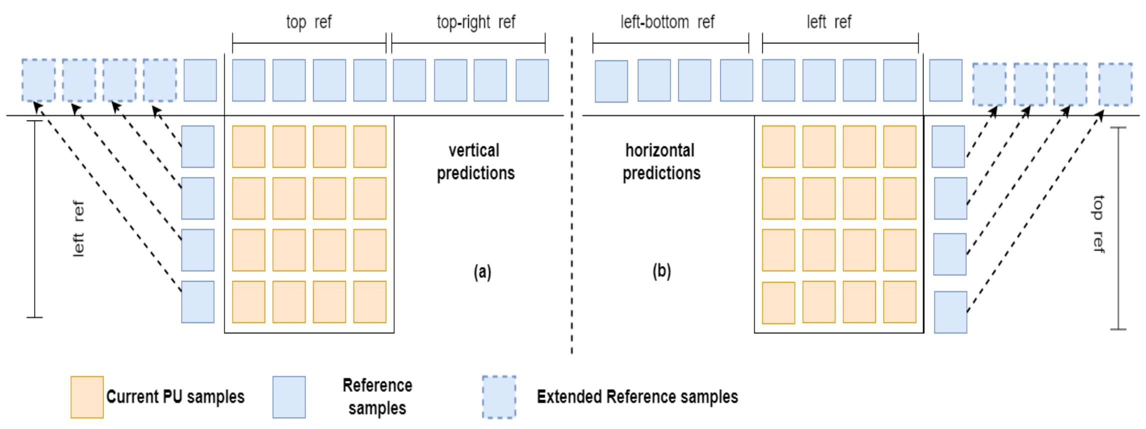


Figure 4.2: Demonstration of reference samples mapping for negative vertical and horizontal predictions in the case of a 4×4 PU. Main array and the extended array are shown. In (b), all the symbols are rotated by 90 degrees in the clockwise direction

In intra module, the neighbouring samples, which are often coded ones, are used as references to predict the current PU block and is shown in Figure 2.7. Each PU is subjected to 35 directional predictions. Modes 0 and 1 are planar and DC predictions, respectively. Directional prediction modes $(2 - 34)$, are grouped into the horizontal $(2 - 18)$, and the vertical $(19 - 34)$ predictions.Steps involved to perform intra predictions is explained in detail in Sections 2.3 and 3.3.1. Planar and DC predictions are coverd in Sections 3.2.1 and 3.2.2 respectively . For directional predictions (vertical or horizontal), the references are grouped as the main array and the side array. In vertical directions, the main array is composed of top-left, top, and top-right samples, with the left column samples used as the side array. Top-left, left, and left-bottom column samples make up the main array for horizontal predictions, with the top row samples as the side array. Only main array samples are used for positive directional predictions, whereas for negative directional predictions, both main and side arrays are used. In negative directional modes, the side array is extended to complete the reference array, and this is demonstrated in Figure 4.2 for a 4×4 PU.

4.3 Hardware architecture of the area efficient DCT/DST based intra prediction engine

The key aspects of the proposed engine are to enable row-wise predictions and operate in a pipelined manner to achieve eight samples throughput with an optimum hardware cost. This section gives the techniques involved to implement the architecture.

4.3.1 Design details of the proposed efficient DCT/DST based intra prediction engine

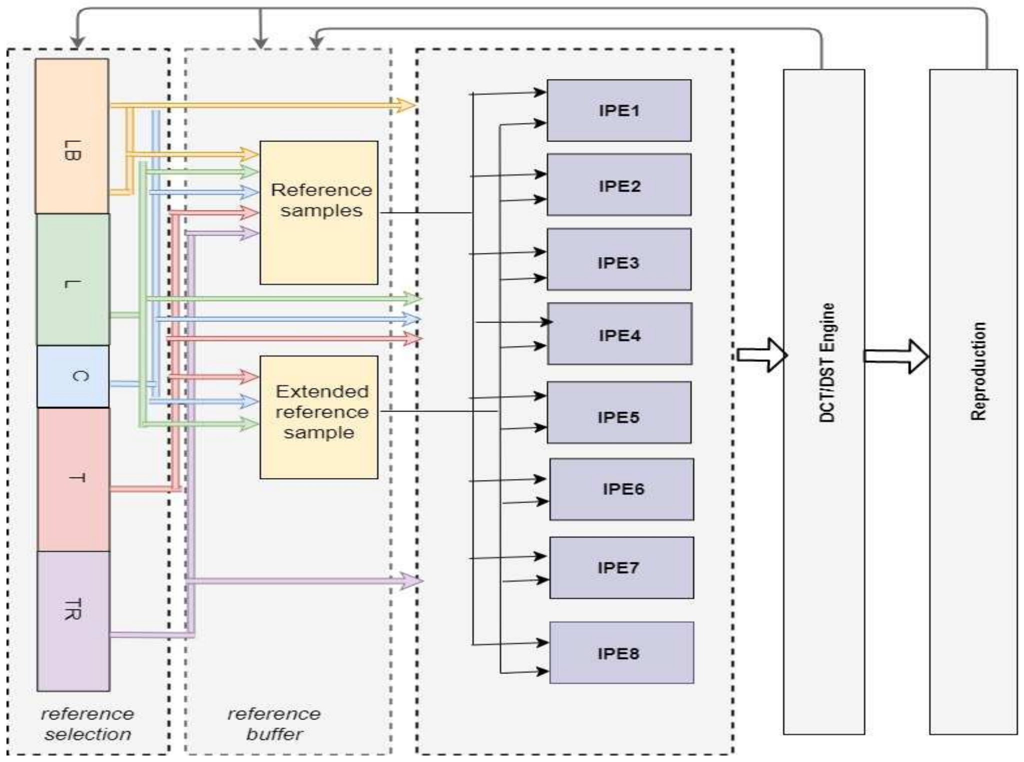


Figure 4.3: Block diagram showing the top-level architecture of the DCT/DST based intra prediction engine

The block diagram of the proposed DCT/DST based intra prediction engine is outlined in Figure 4.3 which consists of the following stages.

1. **Reference selection (RS):** The source references from the neighbouring left, left-bottom, top-left, top, top-right positions of the current PU are fetched for prediction. The availability of the neighbouring references is updated using a

5-bit register.

2. **Reference buffer (RB):** Using the selected references, reference arrays for all the directional modes and PU sizes are pre-determined and stored in registers. Reference samples and extended reference samples modules store the positive and negative directional mode references. For any given directional mode and PU size, the corresponding reference array is chosen and stored in the reconfigurable reference buffer for further processing. Pre determining of the references saves the clock cycles required to fetch them and thus avoid latency associated with data loading.

3. **Intra Prediction Processing Element (IPE):** Intra prediction engine consists of 8 parallel IPEs, which can process 8 samples in one clock cycle. Each IPE has a filtering stage, arithmetic operations unit, prediction unit, as shown in Figure 4.4, and are explained below.

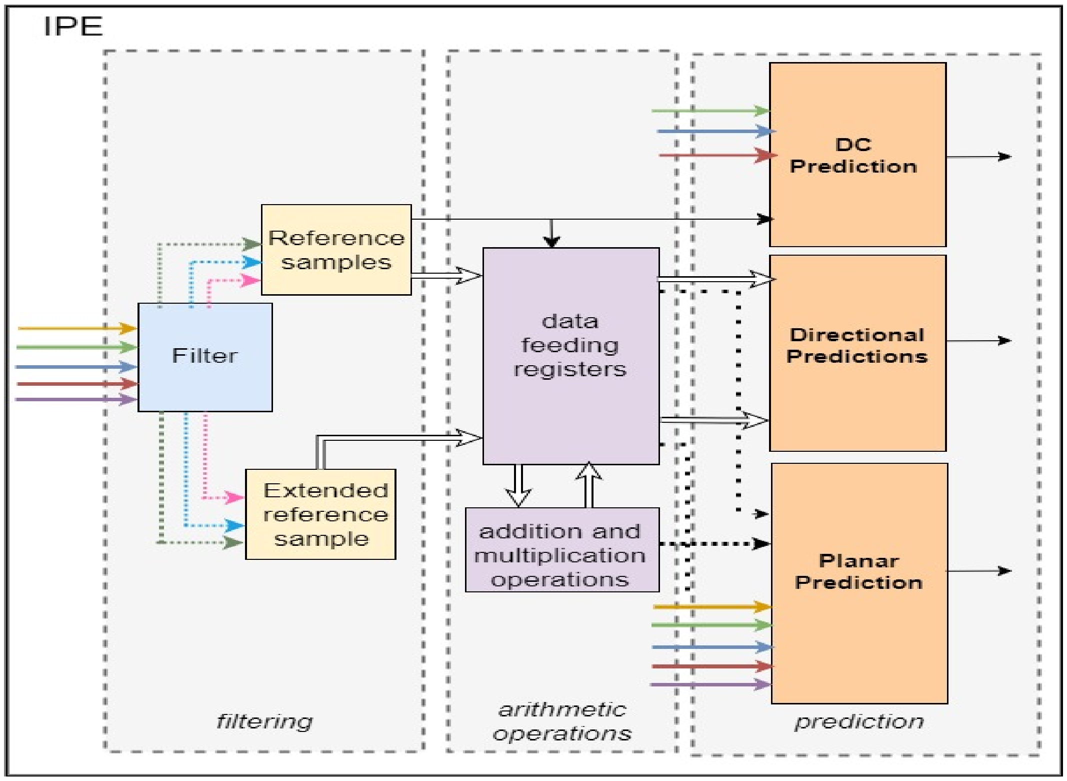


Figure 4.4: Block diagram of the proposed Intra Prediction Processing Element (IPE)

- **Filtering (F):** In this stage, reference samples undergo filtering based on the following conditions. The first decision is to check between three-tap or strong filters based on the threshold value of the sample. Then, based on the directional modes and PU sizes, filtering is applied to the references. The filtered samples move to the next stage. For 4×4 PUs, smoothing is not required. They are moved to the next stage through registers without the need of any filtering.

- **Arithmetic operations (O):** Data feeding registers and Addition-Multiplication (AM) modules are part of this stage. The data feeding module is an intermediate stage solely designed to exchange data between filtering - AM modules and AM - prediction modules. Directional predictions for $2 - 34$ modes are uniform operations, given by (2.10) and (2.11). The addition and multiplication functions in the prediction process are carried out in the AM module. Dedicated AM module reuses the multipliers efficiently and ensures the reuse of DSP slices. The FPGA platform used to test the prediction engine uses DSP48, and the proposed DCT/DST based intra prediction engine uses only forty DSPs.

- **Prediction Unit (P):** Three predictors that processes and produce eight samples is the last stage of the architecture. Each predictor is explained in detail in the later Section 4.3.4.

The proposed DCT/DST based intra prediction engine efficiently processes eight samples in parallel using a pipelined design supporting all the directional modes and the variable sized PUs.

4.3.2 DCT/DST engine based intra prediction unit

The proposed engine is designed to continuously provide unified row-wise predictions. In Figure 4.5, row-wise scanning and selection of PU samples used for prediction are shown. The largest TU is 32×32, with an 8 bit depth video content, a buffer of size 8 KB is required to store the references to predict a current TU/PU. Also, row-wise prediction saves up to 128 clock cycles in case of horizontal predictions, enables DCT/DST and intra engines to operate in parallel to maintain the required high throughput. In addition to this, the hardware consuming intermediate buffer is completely eliminated in the proposed design, which results in an efficient hardware

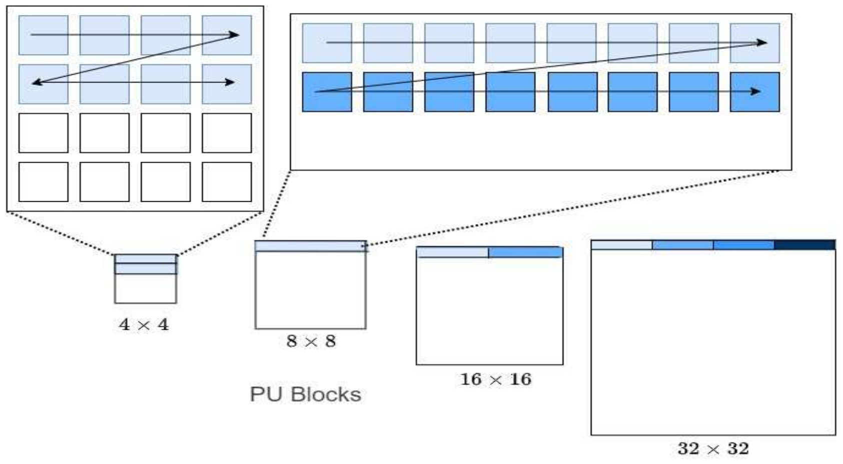


Figure 4.5: Illustration of row-wise prediction of samples in the proposed DCT/DST based intra prediction engine

engine.

4.3.3 Efficient reconfigurable reference buffers

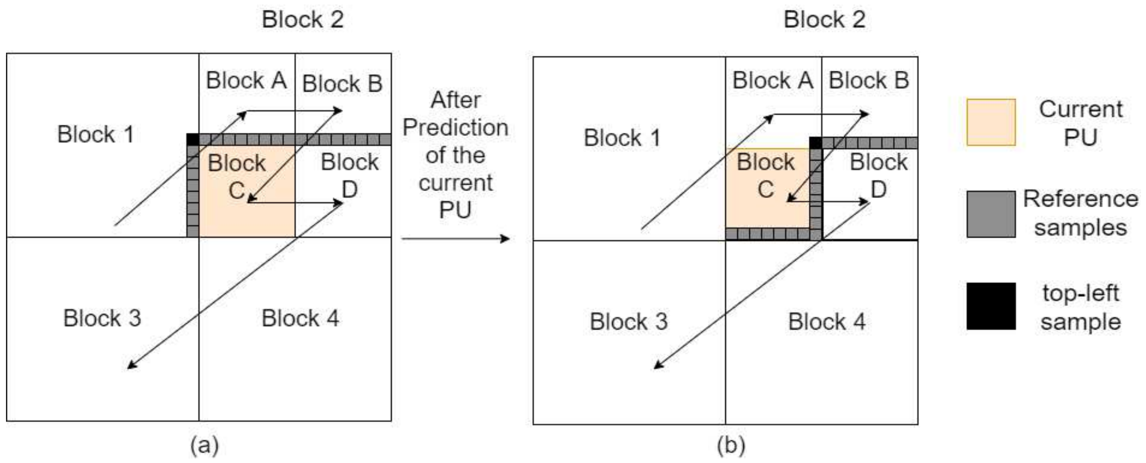


Figure 4.6: Illustration of reference buffers. (a) Reference buffer before the current PU is predicted. (b) Updated reference buffer after the current PU is predicted

Source references are located in five neighbouring positions of the current PU (PU under prediction). Source references are selected in the first stage and are stored in buffers/registers to facilitate the prediction in the second stage. The largest PU used in the implementation is 32×32. Hence in the proposed design, a reference buffer of size 776 bits is incorporated. i.e, the range of references for a 32×32 PU, is limited to

-32 to $+64$ reference samples. For an 8 bit-depth video input, the maximum buffer size required is 776 bits.

The mechanism of reference buffer management is demonstrated in Figure 4.6. The PU and the necessary references used for prediction are highlighted. The PUs are scanned in the Z-order. Block C is the PU under prediction which, is a sub-block of Block 2. Prediction is already completed in Blocks 1, A and B and samples are available to use for prediction and are shown highlighted in Figure 4.6. Border samples from predicted Block C act as the references for predicting Block D. These samples replace the older data in the reference buffer, as demonstrated in Figure 4.6 (b). The same reference buffers are reused to store new data. Some samples get overlapped in the reference buffer, some get replaced, while the unused older samples are discarded. Thus using the reference buffer makes the management and storing of neighbouring samples simpler. Availability and validity of source references is easily determined using the reference buffer. The reference buffer used in the design simplifies fetching operation and saves the clock cycles required to fetch reference samples. For example, a 4×4 PU requires 17 references for prediction. With eight samples processing engine, 3 clock cycles are required to fetch the references. Similarly, PUs of sizes 8, 16, 32 require 33, 65 and 129 references for prediction, which needs 5, 9 and 17 clock cycles respectively to fetch the references. In the proposed design, references are pre-determined and stored in temporary registers, so reading/writing of the references takes just one clock cycle which results in a high speed operation. Also, 776 bits buffer used in the proposed design is the smallest compared to other state-of-the-art designs available in the literature.

4.3.4 Planar, DC and directional prediction architectures to support DCT/DST intra prediction engine

The final stage in the proposed DCT/DST based intra prediction architecture is equipped with three prediction engines, viz. planar, DC, and the directional prediction (for predicting 33 directional modes) modules. The predicted bottom row and the right column samples of the current PU act as the source references for the next PU prediction and gets updated in the reference buffer. All three predictors are designed to process and produce eight samples in each clock cycle.

4.3.4.1 Planar prediction engine

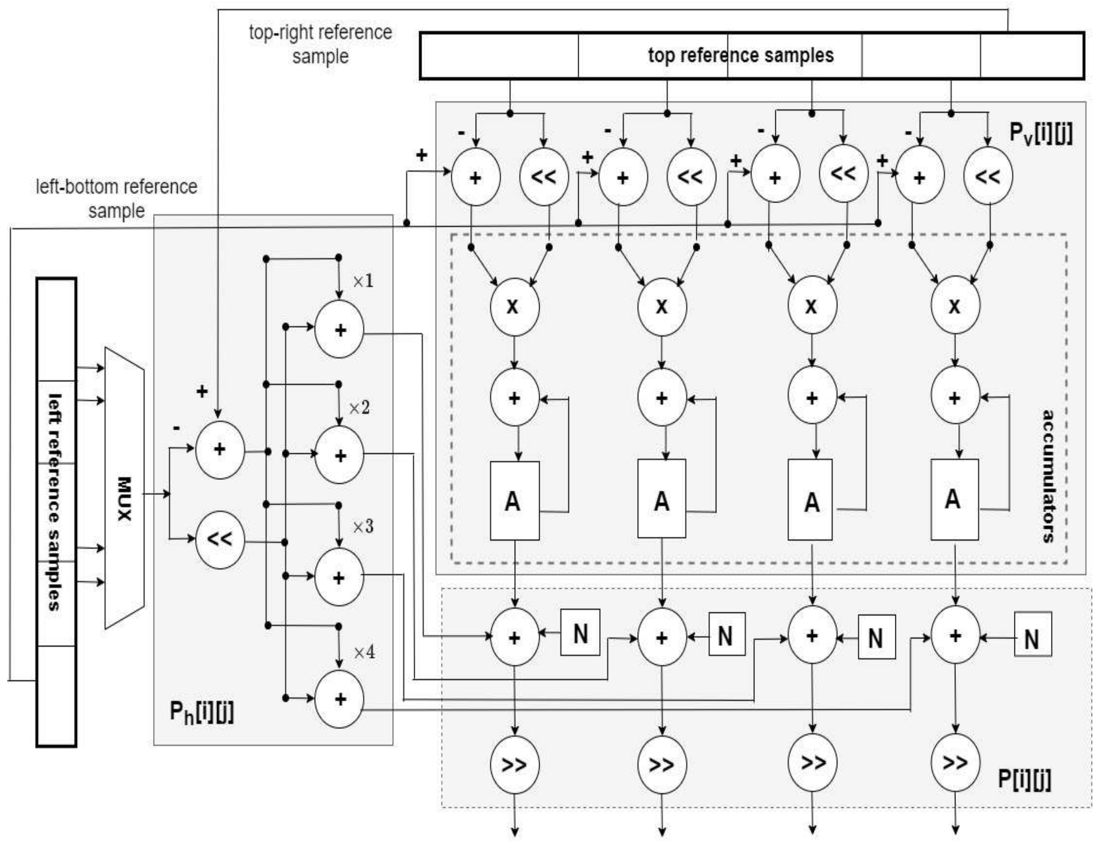


Figure 4.7: Block diagram showing the architecture of planar prediction engine for a 4×4 PU

Block diagram of the planar predictor, for a PU of size 4×4 is shown in Figure 4.7. Sample by sample prediction is carried out by taking the weighted average of horizontal and vertical references. Each sample prediction involves multiplication followed by accumulation of the multiplied results. In the proposed planar predictor, accumulators are used to ensure row-wise predictions during both horizontal and vertical modes.

4.3.4.2 DC prediction engine

DC prediction takes place in two steps:

- Computation of DC_{avg}: The neighbouring top and left filtered references of the current PU are used to compute DC_{avg}, using the relation (3.4).

- Boundary sample filtering and output computation: The decision to filter the boundary samples is made first. If the PU size is 32, then DC_{avg} is the final output. For chroma PUs, DC_{avg} is the final output. For PUs of luma with sizes 4, 8, 16, the final output is the filtered samples from the neighbouring top row and the left column.

A reusable adder module is used for DC prediction. This module comprises four three-input adders. The adder module computes the DC value and is also used for filtering the boundary samples.

4.3.4.3 Directional prediction engine

Directional prediction takes place in three steps, which are as follows:

- Reference samples for each angular predictions are selected in a unique way. For negative modes, side and main arrays are used, which are stored in *extended reference samples* and *reference sample* modules, respectively. For positive modes, only the main reference array is selected. Two references are required to predict one sample and are chosen from the array.
- Index and weights are generated. The index gives the position of the source reference, using which its offset is also found.
- Each sample is predicted using the index - ind, and weight - $fracPrt$.

The engine ensures row-wise predictions by following a mechanism to select reference samples from the filter stage. For the vertical predictions, the position (ind) of its input references sample given by relation (3.9), and its offset ($ind+1$) are determined. Using these two source references, the prediction is derived from (2.11). In the case of horizontal predictions, source reference and its offset are from neighbouring left column, which are not in the same row. Therefore, each prediction sample has to follow its own source selection logic. However, all predictions in the same column will always have the same set of references in the vertical direction. In this case, the reference registers are operated as the shifting window. In the case of each set of horizontal predictions, the projection of each prediction in the same row is maintained constant, and the main registers perform the right shift operation on the sample pattern to complete the row prediction. Index and weights for directional modes for all the PU sizes are fixed. Hence in the proposed design, index and weights for all the directional modes and PU sizes are pre calculated and stored using registers in the design.

Limitations of the proposed DCT/DST based intra prediction engine: Row-wise predictions in the case of planar and horizontal directional predictions takes more cycles to process. The current PU requires neighbouring PU samples for prediction, which results in latency.

4.3.5 Timing analysis of the proposed DCT/DST based intra prediction engine

Intra prediction is fully pipelined and the timing analysis of a 4×4 PUs is demonstrated in Table 4.1. The same analysis holds good for larger size PUs. In Figure 4.6, assume Block A, B, C and D to be 4×4 PUs, and Block 1, 2, 3 and 4 to be 8×8 PUs. Block A is the first PU under prediction considered in the analysis. In $reference\ selection$ *(RS)* stage all the required references for all PU sizes are loaded into the reference registers in one clock cycle. Reference arrays for all prediction modes and all PU sizes, are determined and saved in registers in this stage. In the next clock cycle, based on the prediction mode, corresponding reference buffers *(RB)* are updated.

The intra prediction engine, consists of 8 IPEs in parallel that perform filtering *(F)*, arithmetic operations *(O)* and predictions *(P)* on eight samples in every clock cycle. Hence for a PU of size 4×4, 2 clock cycles are required for each of the above operations. This is demonstrated in Table 4.1, A1 represent processing the first and second rows (8 samples) of Block A. A2 stands for processing third and fourth rows of the Block A - PU. Thus, Block A (A1 and A2) completes filtering in clocks 3 and 4; arithmetic operations in clocks 4 and 5; and predictions in clocks 5 and 6. *RS* operation for Block B starts in clock 3, however *RB* is stalled till Block A is predicted. Once PA1 is completed in clock 5, *RB* for Block B can resume followed by *FB*, *OB* and *PB*. Similarly, as there is a dependency of Block D on the reference samples of Block C, *RB* operation for Block D is stalled till clock 11. Block 3 prediction follows in the similar fashion.

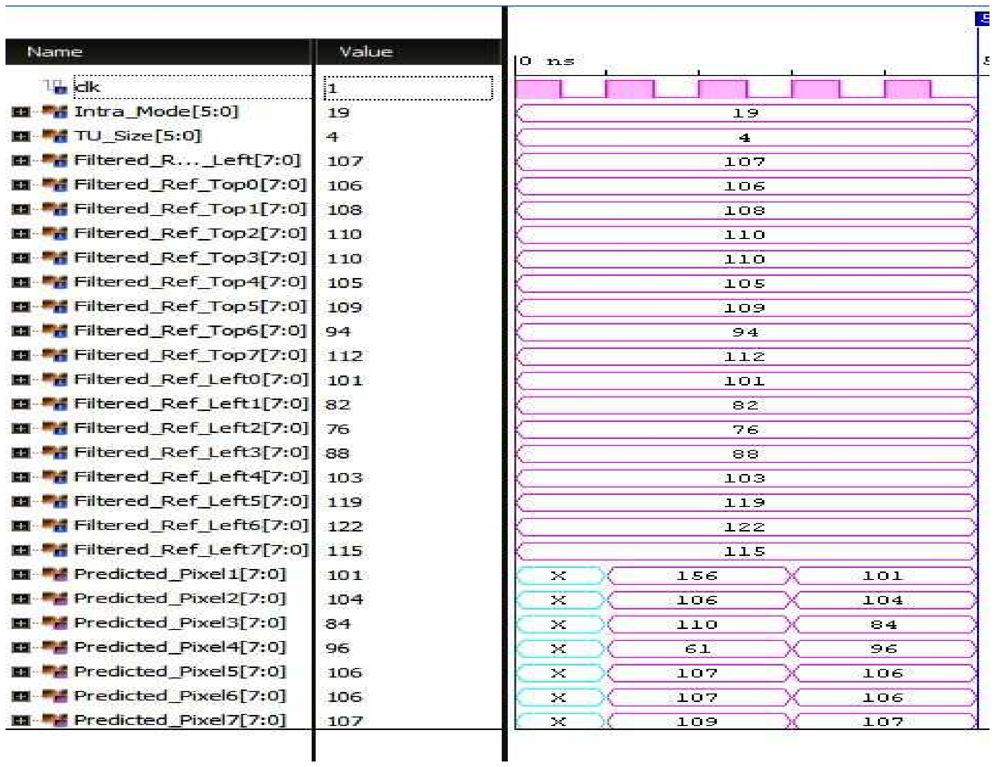
Table 4.1: Timing analysis of the proposed DCT/DST based intra prediction engine

Clock	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Block A1	RS	RB	FA1	OA1	PA1	-	-	-	-	-	-	-	-	-		
Block A2	-	RS	RB	FA1	OA2	PA2	-	-					-	-	-	-
Block B1	-	-	RS	-	-	RB	FB1	OB1	PB1							
Block B2	-	-	-	RS	-	-	RB	FB2	OB2	PB2	-	-	-	-	-	-
Block C1	-	-	-	-	RS	-	-	RB	FC1	OC1	PC1	-	-	-	-	-
Block C2	-	-	-	-	-	RS	-	-	RB	FC2	OC2	PC2	-	-	-	-
Block D1	-	-	-	-	-	-	RS	-	-	-	-	RB	FD1	OD1	PD1	-
Block D2	-	-	-	-	-	-	-	RS	-	-	-	-	RB	FD2	OD2	PD2

RS - Reference Selection *RB* - Reference Buffer *O* - Arithmetic Operated Samples *P* - Predicted Samples

4.4 Experimental results and analysis of the DCT/ DST based intra prediction engine

The proposed DCT/DST based intra prediction engine is implemented on a 28nm Artix-7 FPGA board with a dual-core ARM Cortex-M1 processor to test the performance. The prediction engine is described using Verilog hardware description language (HDL). For analysis, simulation, synthesis, and implementation reports are generated using the Vivado design suite. The behavioural simulation results of angular predic-



Name	Value
clk	1
Intra_Mode[5:0]	19
TU_Size[5:0]	4
Filtered_R..._Left[7:0]	107
Filtered_Ref_Top0[7:0]	106
Filtered_Ref_Top1[7:0]	108
Filtered_Ref_Top2[7:0]	110
Filtered_Ref_Top3[7:0]	110
Filtered_Ref_Top4[7:0]	105
Filtered_Ref_Top5[7:0]	109
Filtered_Ref_Top6[7:0]	94
Filtered_Ref_Top7[7:0]	112
Filtered_Ref_Left0[7:0]	101
Filtered_Ref_Left1[7:0]	82
Filtered_Ref_Left2[7:0]	76
Filtered_Ref_Left3[7:0]	88
Filtered_Ref_Left4[7:0]	103
Filtered_Ref_Left5[7:0]	119
Filtered_Ref_Left6[7:0]	122
Filtered_Ref_Left7[7:0]	115
Predicted_Pixel1[7:0]	101
Predicted_Pixel2[7:0]	104
Predicted_Pixel3[7:0]	84
Predicted_Pixel4[7:0]	96
Predicted_Pixel5[7:0]	106
Predicted_Pixel6[7:0]	106
Predicted_Pixel7[7:0]	107

Figure 4.8: Simulation results of mode 19 angular prediction

tion mode 19 is shown in Figure 4.8. The simulation results shows that eight row-wise samples are predicted in each clock cycle. The experimental results in Table 4.2 summarizes the hardware cost in terms of gate count (LUTs), slice registers and DSPs for each pipeline stage of the proposed engine. The intra top module in the table combines all the five stages into one module. The DCT/DST based intra prediction engine uses 16.2 K LUTs, 5.7 K registers and 40 DSPs, and operating frequency is 150 MHz. Engine supports all PU sizes and all directional modes, and produce an output of 64 samples/clock cycle. The engine processes 8 samples in every clock cycle, each

Table 4.2: Resource utilisation of each pipeline stage of the proposed DCT/DST based intra prediction engine

Module	Sub module	Gate count	Register count	DSP count
Reference selection	-	259	129	-
Reference buffer	-	4268	1199	-
Filtering	-	2879	1047	-
Arithmetic operations	data feeding reg AM	967 755	213 563	- 40
Prediction	DC Planar directional	625 242 5401	109 66 1087	- - -
Intra top	-	802	1360	-
Total	-	16216	5773	40

sample is represented using 8 bits, which results in $8 \times 8 = 64$ samples output. Table 4.3 compares the performance of the proposed DCT/DST based intra prediction engine with other state-of-the-art counterparts in terms of technology, clock speed, supported PU sizes, supported directional modes, hardware cost (in terms of LUTs and registers used by the design), hardware efficiency, supported video applications and the reference buffer size used in the architecture. Hardware efficiency is equal to ($Max\ Frequency \times Througput\ /\ Hardware\ cost$).

In Abeydeera *et al.* (2016), a HEVC decoder is designed, and in Table 4.3 only the intra prediction module parameters are listed. Hardware cost is 63.3% higher, and the buffer memory size is 90% larger than the proposed architecture. Design in Amish and Bourennane (2016), includes five parallel datapaths in order to achieve higher throughput. However, the strategy is limited to 4 K @ 24 fps video applications, with a hardware resource utilisation of 77 K LUTs, using 90.4% more hardware. Design doesn't mention using register buffers and performance is tested on both Virtex- 6 and 7 FPGA platforms.

Table 4.3: Comparison of the proposed DCT/DST based intra prediction engine with other similar works

Parameters	Abeydeera *et al.* (2016)	Amish and Bourennane (2016)	Choudhury and Rangababu (2017)	Min *et al.* (2017)	Ding *et al.* (2019)	Proposed MPPEIP engine	**Proposed DCT/DST based intra prediction engine**
Technology	28 nm	28 nm	28 nm	65 nm	28 nm	28 nm	**28 nm**
Max Frequency	150 MHz	219 MHz	119 MHz	110 MHz		110 MHz	**150 MHz**
PU size	all	all	all	all	all	all	**all**
Directional modes	all	all	all	all	all	all	**all**
Throughput (samples/clock cycle)	2.6	0.2	16	4	16	8	**8 samples of 8-bit depth**
Hardware cost, LUTs	43 K	170 K	112 K	14 K	31.2 K	16 K	**16.2 K**
registers	22 K	110 K	12 K	5.5 K	10 K	122	**5.7 K**
Hardware Efficiency (8-bit video and K LUTs)	9.06	0.25	17	31.4	89.7	55	**74.07**
Video specification	4 K	4 K	4 K	4 K	2 K	4 K	**4 K**
frames per second (fps)	30	24	30	30	60	30	**30**
Ref Buffer size	8 KB	–	–	6 KB	7.6 KB	1 KB	**0.8 KB**

Intra predictor in Choudhury and Rangababu (2017), process 16 samples/cycle. Design is PU based, which requires intermediate memory. The design predicts one 4×4 PU using 6 cycles, which gives an average throughput of 2.33 samples/cycle. The design uses 85.5% more LUTs and 94% more registers than the proposed design. Fully pipelined intra predictor in Min *et al.* (2017) predicts 4 samples in parallel for each clock cycle. LUTs count is 13% less than the proposed design, but the reference buffer memory size is 86.6% larger and throughput is 50% less. Data dependency in intra module is avoided with modified scanning techniques, which produces both row-wise and column-wise prediction samples, and hence an intermediate memory buffer becomes necessary.

In Ding *et al.* (2019), a flexible intra encoder engine is proposed, and in Table 4.3 only the intra prediction parameters are listed. This engine has 17% better hardware efficiency than the proposed design, but at the cost of 48% more hardware and 9.5 times larger buffer size and runs at 16% higher frequency. Also, because of more parallel elements in the design most of the hardware remains unused most of the time. The intra predictor engine in MPPEIP stores the predicted samples in the buffer memory, which is a hardware overhead, and the reference buffer size is 20% larger than the proposed design. Pre determining and storing *ind* and *weights* for all PUs and directional modes result in a higher slice register count in the proposed design, which doesn't affect the hardware efficiency or throughput.

Figure 4.9 and Figure 4.10 shows the graphical representation of the proposed work with other similar state of the art works available in the literature. In Figure 4.9, hardware cost and throughput are compared. The hardware cost of the designs in Abeydeera *et al.* (2016), Amish and Bourennane (2016), (Choudhury and Rangababu (2017)) are significantly higher. Ding *et al.* (2019) and MPPEIP engines uses almost the same amount of hardware. But Ding *et al.* (2019) achieves the same throughput as the proposed engine, but uses 9.5 times larger reference buffer and runs at 16% higher frequency as shown in Figure 4.10. The MPPEIP engine achieves the same throughput as the proposed design, using almost the same amount of hardware but uses 20% larger reference buffer memory. Figure 4.10 clearly shows that the proposed work uses smaller reference buffer memory than other similar works.

The novelty of the proposed DCT/DST based intra prediction hardware architecture is in the elimination of intermediate buffer memory by designing a row-wise intra

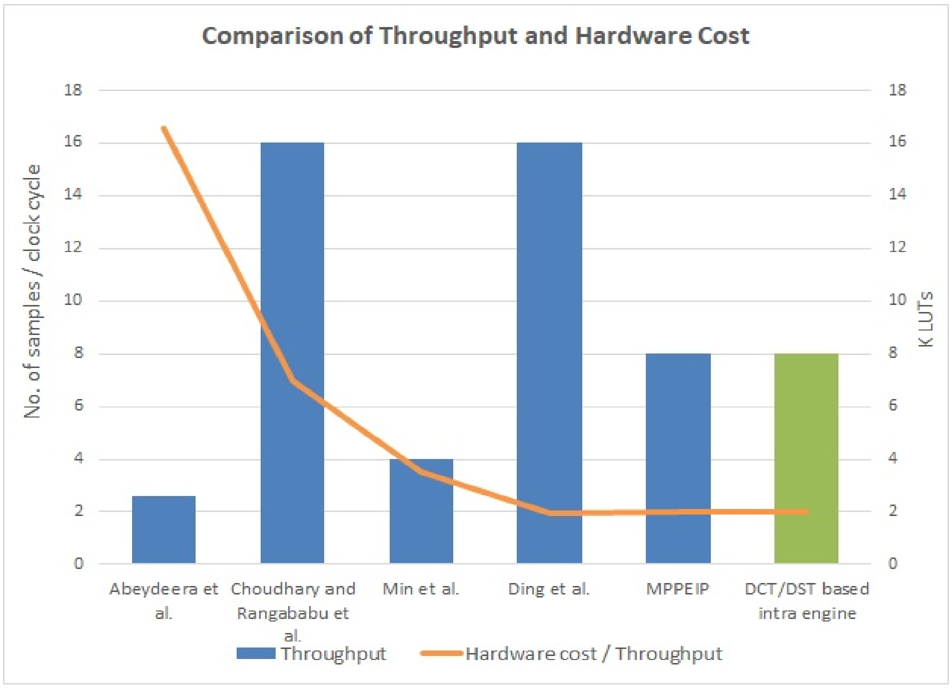


Figure 4.9: Graphical representation of hardware cost and throughput of the proposed DCT/DST based intra prediction engine with other similar works

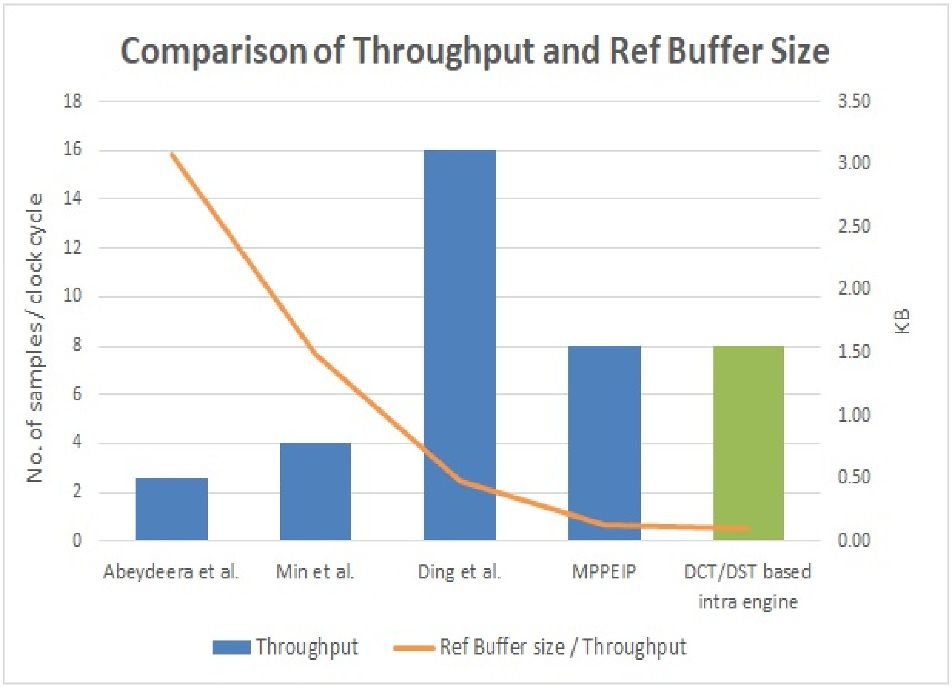


Figure 4.10: Graphical representation of comparison of reference buffer memory size used in the proposed DCT/DST based intra prediction engine with other similar works

prediction engine. This enables the intra and DCT/DST modules to operate in parallel in the HEVC encoder to achieve high throughput. The proposed DCT/DST based intra prediction engine has a smallest reference buffer compared to the other works and achieves a throughput to support HD video applications at a low hardware cost.

4.5 Summary

In this chapter, an improved efficient DCT/DST based intra prediction architecture to support DCT/DST engine in HEVC encoder is proposed. This design supports all PU sizes and 35 directional modes to process 64 samples (8 samples each with 8-bit depth) in parallel in each clock cycle. The proposed DCT/DST based intra prediction architecture is designed using combination of parallel and pipelined techniques that aims to achieve higher throughput. Moreover, the engine is configured to always perform row-wise predictions to maintain the required high throughput and ensures that no extra intermediate buffer memory of 8 K is required to store the predicted samples and saves up to 128 clock cycles in case of horizontal predictions. The compact reconfigurable 0.8 KB reference buffer reduces the data loading delays and hardware cost. A dedicated arithmetic unit ensures the reuse of multipliers to enhance the engine's hardware efficiency. The DCT/DST based intra prediction design uses only 40 DSP slices. Using a 28 nm technology FPGA board operating at 150 MHz, a throughput of 64 samples is achieved with the hardware cost of 16.2 K LUTs and 5.7 K registers to support real time 4 K video encoding.

Chapter 5

Design and implementation of a hardware efficient integrated in-loop filter for HEVC encoder

If four things are followed - having a great aim, acquiring knowledge, hard work, and perseverance - then anything can be achieved.
— A. P. J. Abdul Kalam, Scientist

5.1 Introduction

In HEVC, the in-loop filter is present in both the encoder and the decoder. The in-loop filter consists of a deblocking filter (DF) and a sample adaptive offset (SAO) filter, which helps to improve the subjective quality of the image. The in-loop filter significantly increases the computational load on the HEVC encoder. It is difficult to design an in-loop filter on the hardware that can handle high computations and use the least amount of on-chip memory. The in-loop filter must be able to handle the external memory traffic, dependencies and yet deliver a high throughput to support Ultra HD video applications. In this chapter, an efficient SAO filter that addresses these issues is proposed. This SAO filter is integrated into the in-loop filter and implemented on FPGA. The area efficient architecture implemented on hardware overcomes these latencies and storage overheads to support the real-time high-speed video applications.

The deblocking filter (DF) smooths out blocking artifacts around transform edges in the reconstructed image to improve picture quality (Norkin *et al.* (2012)). The DF is more simple and include parallel operations in HEVC. The SAO filter is designed to improve video quality objectively and subjectively (Fu *et al.* (2012)). H.264 uses macro blocks in all of its modules, whereas in-loop filter in HEVC is picture-based. The input to the in-loop filter is a picture (series of CTUs), where as all other modules use coding tree units (CTUs). Due to the use of distinct processing elements, long delays are caused during encoding and decoding. There is a latency between the DF and SAO filters because the SAO filter uses nearby CTU samples from the DF as references. In DF, vertical edges are filtered first, next the horizontal edges are filtered. Because of this, the edge samples must be stored in memory, which is an overhead causing read-write latencies. There is no data dependency in the band offset (BO) mode of the SAO filter, but it exists in the edge offset (EO) mode. When the DF and the SAO filter are combined, data dependency is a significant factor. The proposed integrated in-loop filter and SAO filter designs employ the following techniques to address the aforementioned issues.

The following are the key contributions of this chapter:

1. An area-efficient DF with high throughput is proposed where transpose data between vertical and horizontal filtering is not stored on, on-chip memory but using registers. The picture data is stored in temporary registers using a novel mechanism that takes advantage of the read address coming from the memory to filter. A relation between the addresses of normal and transposed images is formulated, enabling a transposed image to be loaded directly without the use of an external transpose block.

2. A mixed parallel-pipelined SAO architecture is proposed to support Ultra HD video applications.

The remainder of the chapter is organised as follows: Section 5.2 gives an overview of DF and SAO filters. The implementation of the proposed integrated in-loop architecture is explained in Section 5.3. The detailed performance analysis, and experimental results are discussed in Section 5.4. Finally, conclusions are drawn in Section 5.5.

5.2 Semantics of in-loop filter

In-loop filter in HEVC specifies two modules: a DF that is applied initially, and an SAO filter that is applied to DFs output (Sze *et al.* (2014)). The input video frame is divided into 64×64 LCUs. The LCUs are further subdivided into CUs of size 32×32 down to 4×4 samples in a quadtree form. The CUs are subdivided into prediction units (PUs) and transform units (TUs). Prediction and transform tools are applied on PUs and TUs respectively. A series of 64×64 LCUs/picture is applied as inputs to the in-loop filter.

5.2.1 Deblocking Filter (DF)

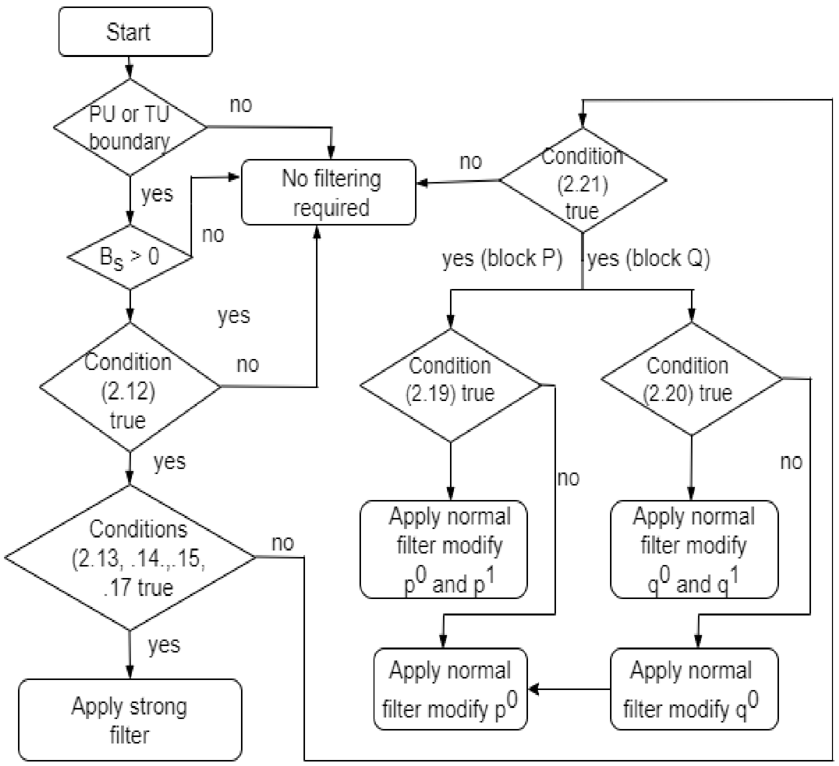


Figure 5.1: Deblocking decisions flow chart (Source: Sze *et al.* (2014))

The DF applies an adaptive smoothing filter to smooth out the discontinuities that arise between the PU and the TU block boundaries in a reconstructed picture frame (Wien (Jan. 2015)). The working of DF is explained in detail in Section 2.4.1. The flowchart shown in Figure 5.1 explains the functional flow of the DF. As seen in the flowchart, several decisions are made in the DF. If filtering needs to be applied or not, and the level of filtering required are decided based on the relations given in Section 2.4.1.

5.2.2 Sample Adaptive Offset (SAO) Filter

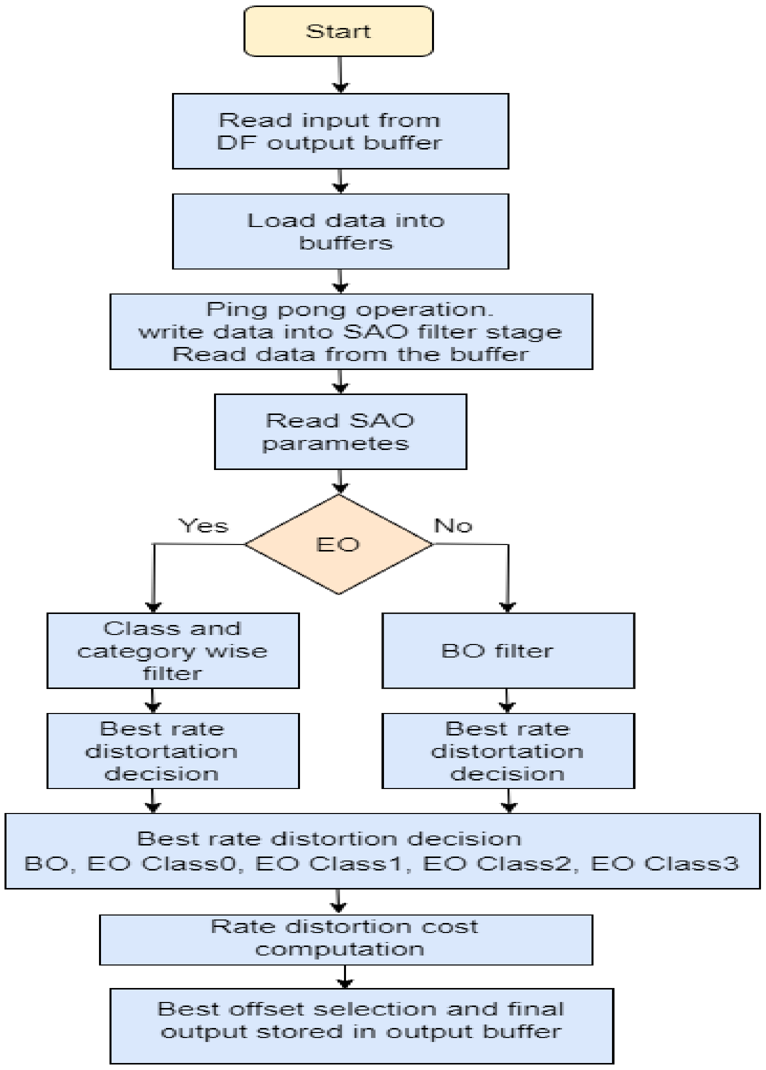

Start
Read input from DF output buffer
Load data into buffers
Ping pong operation. write data into SAO filter stage Read data from the buffer
Read SAO parametes
EO
Yes
No
Class and category wise filter
BO filter
Best rate distortation decision
Best rate distortation decision
Best rate distortion decision BO, EO Class0, EO Class1, EO Class2, EO Class3
Rate distortion cost computation
Best offset selection and final output stored in output buffer

Figure 5.2: Sample adaptive offset filter functional flow chart

The SAO filter in HEVC is CTU-based and applies an offset to the reconstructed pictures to reduce degradation of the reconstructed image (Fu *et al.* (2011)). The use of high QP values in transform module causes ringing artifacts at the edges and this detracts the input image from its subjective quality. The SAO filter reduces ringing effects and thereby reduces the difference between original and restored images. In the

SAO filter, luma and chroma components of the image are filtered in the same way. In SAO, offset is applied using two methods; edge offset and band offset methods, explained in detail in Section 2.4.2. The functional flow of the SAO filter is explained using the flowchart shown in Figure 5.2. Fast rate-distortion method is used to find the best offset in SAO filter (Fu *et al.* (2012)). Encoder decides the best mode among EO, BO or not to apply SAO filtering. The best offset to apply is also decided at the encoder. The method used to generate offset is explained in Section 2.4.2.

5.3 Hardware implementation of the proposed integrated in-loop filter

The top-level architecture of the proposed integrated in-loop filter hardware design is shown in Figure 5.3. The design supports pipeline, parallel, and sequential techniques. The *DF*, *SAO* filter, an *in-loop filter scheduler* are the main functional modules in the architecture. The functional flow of the DF is shown in Figure 5.1. The *in-loop filter scheduler* is responsible to generate all of the control signals required to transmit input data to the *DF*, manages data transfer between the *DF* and the *SAO* filters using the intermediate *DF buffer*, and finally assists in the storage of the output from the *SAO* in the *output buffer*.

The 8×8 block boundaries are filtered using *horizontal filter (HF)* and *vertical filter (VF)* which are the main functional units in the *DF* module. Each of the *VF* and *HF* modules has four *edge filters* that can process four samples simultaneously. The control unit generates the signals required for operating the *HF* and *VF* modules, such as select, enable, reset, end, etc. In the *average buffer*, the outputs from *VF* and *HF* are averaged before being saved in the *DF buffer*.

The SAO module is made up of three important parts: a *mode decision (MD)* unit that is used to determine the best mode, an *EO*, and a *BO* functional modules. The *EO* and *BO* modules receive an input from the *DF buffer*, which is divided into blocks of 8 samples. The *sample difference (SD)* module generates the difference between the original and restored samples, whereas *EO* and *BO* generate the offset and perform the filtering processes. The distortion calculation is carried out in the *MD* module to calculate the ideal offset based on the offset information and the sample difference, and the best mode for SAO is chosen. The output is populated in the *output buffer*.

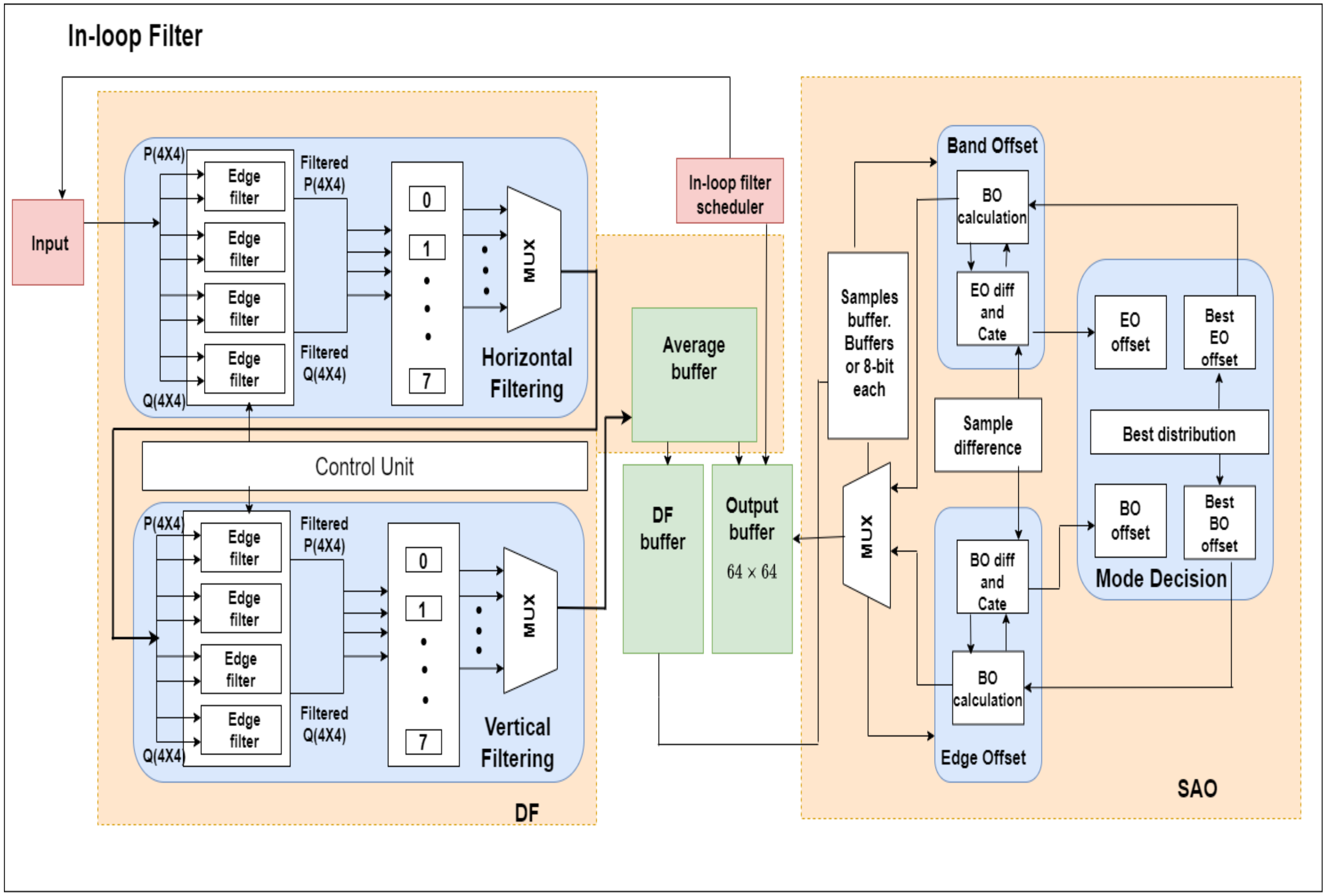


Figure 5.3: Block diagram of the integrated in-loop filter (with DF and SAO filter)

5.3.1 Hardware design of the deblocking filter (DF)

The proposed hardware design of the DF is shown in Figure 5.3, which includes the *control unit*, *VF*, and *HF* modules, as well as *average* and *DF buffers.* The *VF* and *HF* units work in parallel to filter data using edge filters and conditional datapaths, which are detailed in Section 2.4.1.

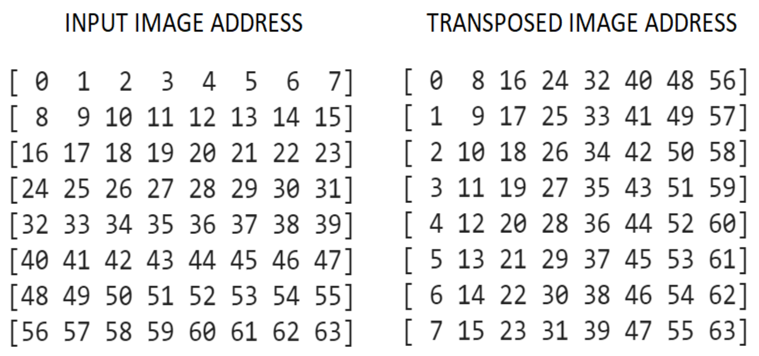
5.3.1.1 Vertical and horizontal filters

The general order followed in DF implementation is to first perform *HF*, then store the data, transpose the data, and finally apply *VF*. This method necessitates the use of a memory block to store the samples before and after the transpose operation, which is an overhead. Additionally, at any given time, only one unit either *HF* or *VF* can operate, reducing hardware efficiency. To address this issue, a novel feature is included in the proposed integrated in-loop filter design, where both *HF* and *VF* operate in parallel. To load the input image data into the *input* memory, a new approach is used, allowing the transposed image to be loaded directly into temporary registers without the use of an external transpose block.

To load an image from memory, the usual method is to first create an address variable, then iterate from top to bottom and read data from each address location, then apply *HF*. Data is stored in a buffer memory, transpose the data, and then repeat the process, finishing with *VF*. The same concept is used in the proposed design, but data for both normal and transposed images are selected at the same time. By formulating the relationship between the original image and its transpose, the read address of the input image is used to obtain the address of the transposed image. The transposed address for the 8 $\times$ 8 blocks is generated in the *control unit*, shown in Figure 5.3. These samples are sent to both the *HF* and the *VF* at the same time.

Figure 5.4 shows a 8 $\times$ 8 image and its transpose. The relationship between the original and transposed image addresses is given by relations (5.1) and (5.2), where F is the LSB and A is the MSB. An 8 $\times$ 8 image has 64 samples and thus requires a 6-bit address. To formulate the relationship between normal and transposed images, a 6-variable K-Map is used and manually solved. The results are shown in (5.3).

$$\text{Input image address} = \{A, B, C, D, E, F\} \tag{5.1}$$



```
INPUT IMAGE ADDRESS              TRANSPOSED IMAGE ADDRESS

[ 0  1  2  3  4  5  6  7]     [ 0  8 16 24 32 40 48 56]
[ 8  9 10 11 12 13 14 15]     [ 1  9 17 25 33 41 49 57]
[16 17 18 19 20 21 22 23]     [ 2 10 18 26 34 42 50 58]
[24 25 26 27 28 29 30 31]     [ 3 11 19 27 35 43 51 59]
[32 33 34 35 36 37 38 39]     [ 4 12 20 28 36 44 52 60]
[40 41 42 43 44 45 46 47]     [ 5 13 21 29 37 45 53 61]
[48 49 50 51 52 53 54 55]     [ 6 14 22 30 38 46 54 62]
[56 57 58 59 60 61 62 63]     [ 7 15 23 31 39 47 55 63]
```

Figure 5.4: Address matrix of the input image and the transposed image

$$\text{Tranasposed image address} = \{A^T, B^T, C^T, D^T, E^T, F^T\} \tag{5.2}$$

$$\begin{aligned} F^T = C, \quad E^T = B, \quad D^T = A, \\ C^T = F, \quad B^T = E, \quad A^T = D \end{aligned} \tag{5.3}$$

The 6-bit variable is iterated from 0 to the image's last address and used to create a new address with the above-shown relation to obtain its transposed addresses directly. The *control unit* employs this mechanism. This eliminates the need for a separate transpose block, reducing memory overhead. By operating *HF* and *VF* in parallel, hardware efficiency of the design is improved. Filter processing time is reduced as read-write latencies that arise due to storing data in transpose buffers is eliminated.

5.3.1.2 Datapath overflow and underflow solution

The conditional datapaths used to make decisions in the DF uses a lot of additions and subtraction operations due to which filter selection functions are prone to overflows and underflows. This poses as a major problem which continues throughout the image calculations and is exacerbated when the corner values of the input samples are evaluated using (2.13) and (2.14). Overflow and underflow produce an extra output bit; if this extra bit is not handled appropriately, the conditional datapaths produce incorrect decisions. The registers used in the hardware to store the samples are 8-bit wide, storing this one extra bit requires the use of one extra register, which is an overhead. This overhead issue is much larger, when the number of decision operations required for one picture frame is taken into account. Handling this extra register requires more conditional evaluations for proper operation. This method is complex and inefficient in terms of memory and area requirements.

This issue is addressed in the proposed design by lowering the brightness of the input image. The average of the input image is taken, and this is used for processing. This allows to use the eighth bit of the same register to store the overflow and underflow bits.

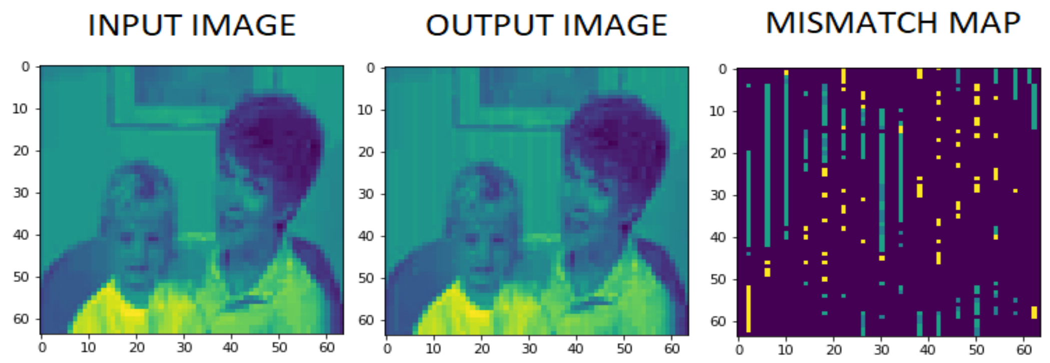


Figure 5.5: Input image, output image, and mismatch map to visualize underflow and overflow problem

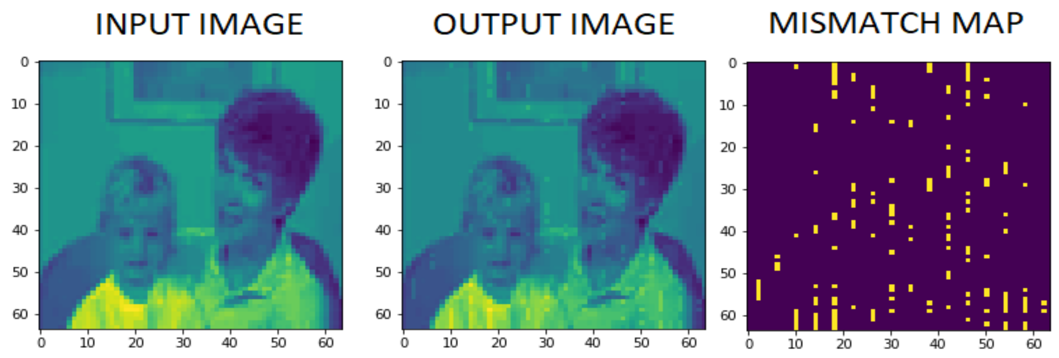


Figure 5.6: Input image, output image, and mismatch map where underflow and overflow problem is solved

The input image is processed by the *DF*, and the results are shown in Figure 5.5. Although the overflow bit is saved, the signing conventions were ignored. Due to this, the conditional datapaths make incorrect decisions. The input image, the output image of the *DF*, and the mismatch map is ahown in Figure 5.5. The mismatch map is the difference between the output and the input images. Mismatch map in Figure 5.5, shows the disparity due to the incorrect decisions. Then the average of the input image is taken, this reduces the brightness of the image, and it can be represented using 7-bits. The overflow bits generated are stored as the 8th bit and the same registers is used in decision making operations. The results of the image with reduced brightness are shown in Figure 5.6, where the mismatch is significantly reduced without the use of additional registers or extra calculations. In the case of Figure 5.6, correct filtering decisions were made in the conditional datapaths. However, the output image shows visual irregularities in the filtered regions. Section 5.3.1.3 explains how this problem is addressed.

5.3.1.3 The over-brightness problem

The DF includes a clipping operation to deal with excessive brightness, clipping alone did not produce satisfactory results, particularly with low-brightness input images. This problem is solved in the proposed design by averaging the input and output images. The effects of excessive brightness are significantly reduced in the filtered areas of the image, and this is shown in Figure 5.7. The visual irregularities in the output image is reduced compared to the results in Figure 5.6, despite the fact that the mismatch map is the same.

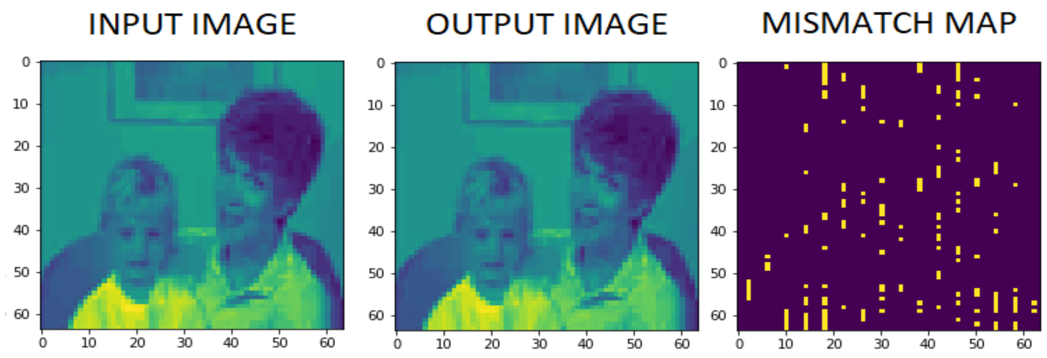


Figure 5.7: Input, output images and mismatch map where excess brightness problem is solved

5.3.2 Hardware implementation of the SAO filter

The proposed SAO filter architecture is shown in Figure 5.3, and the functional flow is explained using the flowchart shown in Figure 5.2. The *DF* output is selected as 8 × 64 sample sets, which are then buffered, processed, and filtered. *EO* and

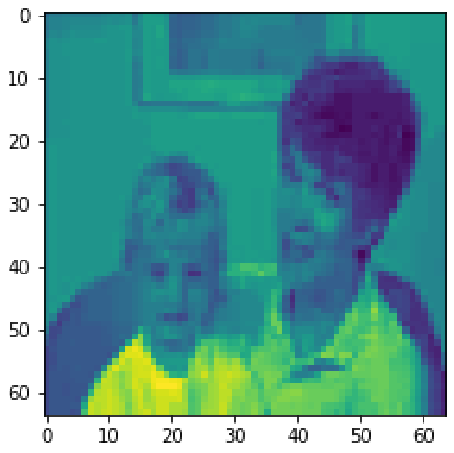
BO is applied on all samples within each LCU block in the *SAO* module which is a computationally intensive process (Wien (Jan. 2015)), (Bossen *et al.* (2012)). The *SAO control* unit is clock-controlled and performs sequential operations for the *SAO* module. The sample sets are computed in parallel to improve overall throughput. The input data flow is properly regulated by accessing 8 sample sets from the buffer in an incremental order by generating read addresses. After reading the input data and storing it in the buffers, the sample sets are processed in a pipelined fashion until all of the data sets are processed. This procedure is carried out for each sample set.

Data dependency is a significant factor when the DF is combined with the SAO filter. In BO mode, no data dependencies exist, whereas in EO mode, there is a data dependency between the current CTU and its neighbouring CTUs. These factors are considered in the hardware design. As shown in Figure 5.3, the *SAO* filtering is made up of *SD*, *EO*, *BO*, and *mode decision (MD)* modules, and their functions are as follows. The *SD* module is responsible for calculating the difference between the restored and original samples. Edge offsets are calculated by sending data (sample) blocks to the *EO* module, where the class and category information from each sample is used to generate offsets. Samples that have been restored are sent to the *BO* module, which generates band position information for each sample. The *SD* module collects sample data and generates differences based on the band position information. The offset is generated after receiving the sample differences from the *EO* and *BO* modules, and the optimal offset type is chosen in *MD* using the rate-distortion method.

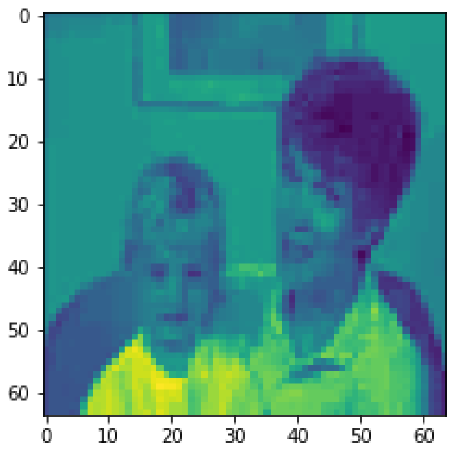
Each EO class and category is processed in the *EO* module and compared with their adjacent neighbours when required, as shown in Figure 2.15. For example, for the EO class 0 category, the horizontal adjacent neighbours are picked and processed. Similarly, in the case of 45°, 90°, or 135° class operations, requires neighbouring samples are selected and offset is determined. In each clock cycle, the *EO* block filters 8 samples. The neighbouring samples are stored in a memory buffer. In one clock cycle, the *BO* module processes 8 samples. Because there is no data dependency in this block, sample preservation is not required, and thus no buffer is used to store the filtered samples. The offsets are generated in the *MD* module, the best mode is chosen, and the output is saved in the *output buffer*.

Limitations of the integrated in-loop filter: Intermediate storage buffer is used between DF and SAO. Edge offset requires neighbouring samples for processing which are stored in buffers, which is an overhead increasing latency.

5.4 Experimental results and analysis of the SAO filter and integrated in-loop filter



(a) Input image



(b) Output image

Figure 5.8: Input and output images from the integrated in-loop filter

The input and output images obtained from the proposed integrated in-loop filter engine are shown in Figure 5.8. The outcome is compared using two quality metrics: structural similarity index (SSIM) and peak signal-to-noise ratio (PSNR). SSIM values range from 0 to 1, with 1 indicating perfect match. The recorded SSIM of the output image from the proposed design is 0.99. The PSNR value recorded is 48.28 db which is very close to the ideal values.

The proposed in-loop filter contains both DF and SAO modules, which can be operated separately or together. More DF hardware implementations are available in the literature, so comparison of the DF filter with other similar works is presented in Table 5.1. In Table 5.1, several ASIC and FPGA implementations of the DF are compared using the following parameters: device (ASIC/FPGA), operating frequency, technology used for the implementation, LCU size considered in the design, silicon footprint in terms of hardware cost (gates in the case of ASICs and LUTs in the case of FPGAs), number of cycles taken for processing one LCU, resolution video supported and finally number of frames per second. The proposed DF produces a throughput of 8

Table 5.1: Comparison of the proposed DF with other similar works

Parameter	Ozcan *et al.* (2013)	Ozcan *et al.* (2013)	Shen *et al.* (2013*b*)	Diniz *et al.* (2015)	Diniz *et al.* (2015)	Peesapati *et al.* (2017)	Peesapati *et al.* (2017)	Kopperundevi *et al.* (2022)	Proposed DF filter
Device	FPGA	ASIC	ASIC	ASIC	FPGA	ASIC	FPGA	FPGA	FPGA
Frequency (MHz)	108	108	200	200	140	322	120	100	200
Technology	Virtex 6 28 nm	CMOS 90 nm	CMOS 130 nm	CMOS 45 nm	Virtex 6 28 nm	CMOS 28 nm	Virtex 6 180 nm	zybo	virtex 6 28 nm
LCU size	64×64	64×64	32×32	64×64	64×64	32×32	16×16	32×32	64×64
Hardware Cost: LUTs	5.2 K	16.4 K	21 K	3.3 K	1398	865 K	341.78 K	3.8 K	1.8 K
No. of Slice registers	1547	-	-	-	441	-	5891	-	1204
Cycles	7680	7680	110	1027	1027	31	22	16	55
Resolution supported	2 K	2 K	4 K	4 K	4 K	8 K	8 K	8 K	8 K
frames per second(fps)	30	30	60	60	60	200	200	90	40

samples per clock cycle. As seen in Table 5.1, it is clear that the proposed DF uses the least hardware and supports ultra HD video applications.

Table 5.2 summarises the performance comparisons of the proposed integrated in-loop filter design with other designs available in the literature. The proposed in-loop filter produces a throughput of 8 samples per clock cycle. Table 5.2 shows the comparisons in terms of device, technology, frequency of operation, supported LCU size, number of cycles required for processing one LCU data, hardware cost (in terms of LUTs and registers used in the design), throughput, and hardware efficiency (HWE). HWE is used as a metric for a common and fair comparison with other works as there is no common metric available. Higher the HWE value, better is the design. The HWE is generated using the following relation:

$$HWE = \frac{\text{Supported resolution } \times fps}{\text{Hardware Cost}} \tag{5.4}$$

A pipelined in-loop filter is designed for an HEVC decoder and implemented on an ASIC platform in Zhu *et al.* (2013*b*). ASIC implementation gives the benefit of a higher frequency. But the hardware cost is 300.8% higher, resulting in an HWE of 26.04, thus making the proposed integrated in-loop filter design 6 times more efficient. The in-loop filter designed and tested on the ASIC platform supports ultra HD video applications in Shen *et al.* (2016). However, the proposed design uses 50% less hardware, resulting in a better HWE. The design in Park *et al.* (2016), tests DF and SAO separately. Each module is tested at two different frequencies. The results of both modules being processed together are not mentioned.The closest operating frequency match with this proposed work is considered and the corresponding throughput is compared. In this case HWE of the proposed design is 47% better even though it is operating at 73.3% lower frequency.

The architecture is implemented with reconfigurable processors in Liu *et al.* (2017). This design's processing capacity is higher but at the expense of extremely high hardware utilization, which reduces its HWE. To achieve higher throughput, Baldev *et al.* (2018), implements a pipelined-parallel in-loop filter, but the LCU size considered in the design is smaller. Furthermore, the hardware cost is nearly eight times higher, and HWE is 144.43% less than the proposed design. A power efficient integrated DF and SAO filter architecture in Singhadia *et al.* (2021), supports all CTU sizes and the design is implemented on both ASIC and FPGA. In Table 5.2 only FPGA implemen-

Table 5.2: Comparison of the proposed integrated in-loop filter (combined DF and SAO filter) with other similar works

Parameters	Zhu *et al.* (2013*b*)	Shen *et al.* (2016)	Park *et al.* (2016)	Liu *et al.* (2017)	Baldev *et al.* (2018)	Singhadia *et al.* (2021)	Proposed integrated in-loop filter
Device	ASIC & FPGA	ASIC	ASIC	RPU	FPGA & ASIC	ASIC & FPGA	FPGA
Technology	65 nm	65 nm	90 nm	65 nm	90 nm	28 nm	28 nm
Frequency (MHz)	240	182	1000/260/970/260	250	366.96	177.82	200
LCU size	16×16	64×64	32&64	64×64	32×32 & 16×16	32&64	64×64
Cycles	16	558	40 & 266	9582	43	16	55
Hardware Cost	31 K gates	103.3 K gates	262.7 K gates	5.4 million gates	593.32 K gates	32.8 K LUTs	7.7 K LUTs
Resolution supported	7680×4320	7680×4320	7680×4320	1920×1080	7680×4320	7680×4320	7680×4320
fps	120	40	771/200/450/120	52	200	46	40
HWE (10^3)	**26.04**	**12.84**	400.37/**143.70** 94.79/**29.72**	**2.96**	**1.118**	**46.52**	**171.61**

tation values are updated. The design uses less number of clock cyles, but at the cost of 76.4% more hardware, due to which the efficiency is 72.8% less than the proposed integrated in-loop filter architecture.

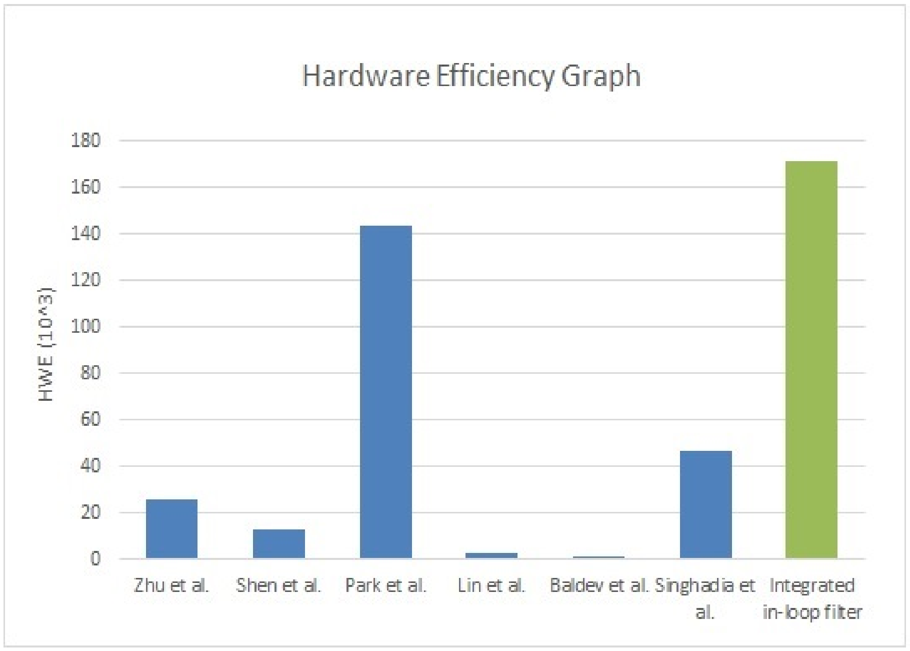


Figure 5.9: Graphical representation of comparison of hardware efficiency of the proposed integrated in-loop filter engine with other similar works

The HWE of the proposed integrated in-loop filter engine and other works (from the Table 5.2) are shown in Figure 5.9. From the graph, it is clear that the proposed integrated in-loop engine outperforms in terms of hardware efficiency when compared to other hardware implementations. With an HWE of 171.61, the proposed design has the best hardware efficiency. The proposed design is the most optimised architecture and produces high throughput, making it highly suitable for portable consumer devices
.

The graph in Figure 5.10 compares the HWE and hardware area used by the proposed integrated in-loop filter design with architectures in Zhu *et al.* (2013*b*), Shen *et al.* (2016), Park *et al.* (2016), and Singhadia *et al.* (2021). The hardware area depicted in the graph is rounded to the nearest thousand. The area utilised by the designs in Liu *et al.* (2017), and Baldev *et al.* (2018) is very high and their HWE is very low, they are omitted in Figure 5.10. The graph in Figure 5.10 shows that the proposed design is the most efficient and uses the least amount of hardware.

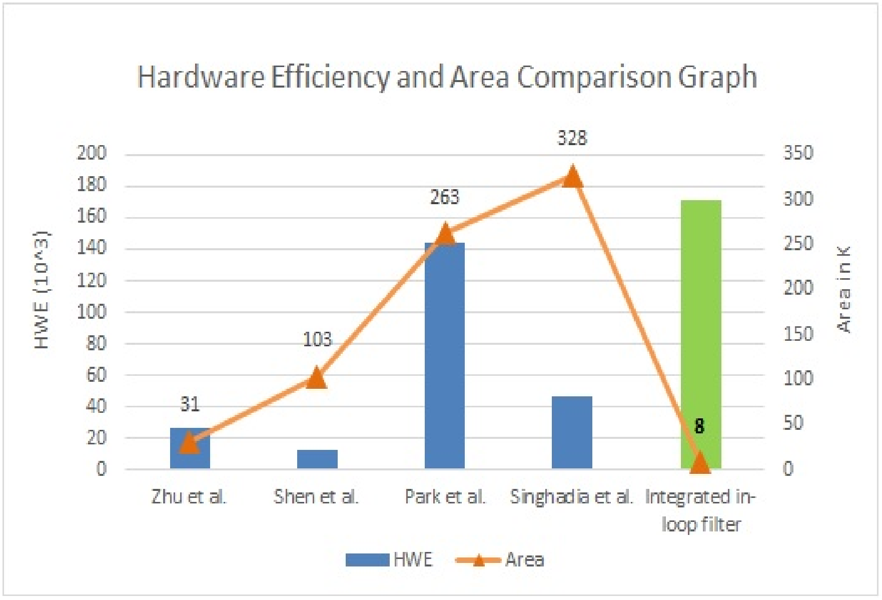


Figure 5.10: Graphical representation of comparison of hardware efficiency and hardware area utilized by the proposed integrated in-loop engine with other similar works

5.5 Summary

Area-efficient hardware architectures of the DF and SAO filters for a HEVC encoder are implemented on FPGA. The proposed design includes a balanced parallel-pipeline scheme that can process 7680 × 4320 videos @ 40 fps, clearly supporting Ultra HD video applications. The design is optimised, using only 7.73 K LUTS and 2.8 K slice registers, which makes it suitable for low-power real-time video applications. The proposed area-efficient reprogrammable in-loop filter is well suited to support higher resolution video applications and suitable for portable consumer devices. The PSNR and SSIM values, at 48.28 dB and 0.99, respectively are close to the ideal values, showing the ability of the design to deliver quality output.

Chapter 6

Conclusion and future scope

Think left and think right, and think low and think high. Oh, the thinks you can think up if only you try!

— Dr. Seuss, Author

6.1 Conclusion

A dedicated HEVC hardware is very useful for many high quality video applications, especially they are suitable for on-the-go video consumption. In the area constrained devices like laptops, cellphones, and dedicated streaming devices, the proposed HEVC hardware can be integrated into the encoder to efficiently encode a high definition video stream. The architectures described in this thesis achieve high throughput with low latencies while using a very low silicon area. The main objective of this research work is to design area-efficient hardware architectures of the intra prediction engine and the integrated in-loop filter for the HEVC video encoder, that supports HD and UHD video applications.

The proposed PDA and PPA architectures perform planar and DC predictions to produce a throughput of eight samples in every operation cycle. The experimental results for PUs of size 4, 8, 16 and 32 are tabulated and compared in terms of number of LUTs and registers used by the hardware architectures. There is a decrease of 20% in LUT consumption in PDA, compared to PPA configuration for 4×4 PUs. The PPA engine uses 20%, 46% and 62% less resources for 8, 16 and 32 PUs respectively than the PDA design. Results show that PPA architecture supports all PU sizes and produce high throughput, hence better suited for area constrained applications. While PDA achieves better results only in the case when the input video frames have a large number of 4×4 PUs. The PPA architecture is extended to support all the angular intra predictions as it produces better throughput using the optimum silicon area regardless of the PU size.

The MPPEIP engine is designed to produce eight samples in parallel in every clock cycle. Techniques such as reusable reference buffers, manual assignment of directional predictions for all PU blocks, and use of a dedicated arithmetic operations unit are implemented in the design. All these features contribute to a significant reduction in hardware resources. A compact reusable reference buffer of size 1 KB is used in the MPPEIP engine. Data loading delays are reduced and the design uses only 40 DSP slices. The experimental results show that the proposed MPPEIP design uses 16 K LUTs and 122 registers of hardware resources which is significantly less hardware than the majority of the existing designs.

An improved efficient intra prediction architecture is designed to support DCT/DST engine in HEVC. The proposed architecture achieves higher throughput by combining parallel and pipelined mechanisms. Techniques such as row-wise predictions that completely eliminate 8 KB intermediate memory buffer; an improved compact reconfigurable buffer of just 776 bits; a dedicated arithmetic unit that uses only 40 DSP slices, are used, which enhance the hardware efficiency of the engine. The experimental results show that the design uses 16.2 K LUTs and 5.7 K registers to produces a high throughput. The proposed DCT/DST based intra prediction engine has low hardware cost, uses the smallest reference memory buffer compared to all the other works, and achieves a throughput to support HD video applications.

An area-efficient hardware architecture of the SAO filter is designed and integrated into the in-loop filter of the HEVC encoder. The design is a balanced parallel-pipeline scheme that supports UHD applications. The hardware cost of the architecture with 7.73 K LUTS and 2.8 K slice registers is the lowest compared to the other state-of-the-art designs, which makes it suitable for low-power real-time video applications especially for portable consumer devices. The experimental results show that the proposed in-loop architecture has the best hardware efficiency (171.61×10^3) compared to other works. The PSNR and SSIM values at 48.28 dB and 0.99, respectively are close to the ideal values, showing the ability of the design to deliver quality output.

6.2 Future scope

In this thesis, area-efficient hardware architectures for two modules (intra prediction and in-loop filter) of HEVC encoder are deigned and implemented on FPGA to support HD and UHD video applications. A balanced parallel-pipeline scheme is used to achieve the research objectives. With the high demand for dedicated HEVC hardware, area-efficient architectures are always sought after.

- The balanced parallel-pipeline and sequential technique used in this thesis can be extended and merged to design an entire intra prediction based HEVC encoder.
- The methodologies presented in this thesis can be extended to design and implement the hardware architecture of the versatile video coding (VVC)/H.266 codec.
- Input image/video stream used for testing the FPGAs is mapped to an SD Card and HDMI input and is not possible to modify in real-time. This is a limitation. The inputs used to test the architectures are also pre-generated. It would be interesting and challenging to test the HEVC modules with the real-time input data.
- The degree of parallelization adopted in the design is set by the throughput required to support certain video applications. Loosening this constraint would lead to other optimal solutions.

Appendix I
FPGA board details

Searching for new ideas is an endless process.

— R. K. Laxman, Cartoonist

A field-programmable gate arrays (FPGAs) are reprogrammable semiconductor ICs, that are made up of a matrix of configurable logic blocks (CLBs). FPGA's contain large number of CLBs that are connected via programmable interconnects. In addition to CLBS there are lookup tables, flip-flops, shift registers and other elements which can be configured to perform various logic functions. FPGAs are dynamically reprogrammable and suitable for various applications like video processing, data analytics, image inference, encryption, compression etc. Optimized FPGAs are power-efficient and faster than running equivalent workloads on a CPU. In short, FPGAs are versatile, efficient, and offer better performance. All these features makes FPGA good choice for implementing the HEVC's intra prediction and in-loop filter architectures. In this section, the characteristics of the FPGA used in this work, and other interfaces used in the design are discussed.

Modern FPGAs consist of up to two million logic cells that can be configured to implement a variety of algorithms. The basic structure of an FPGA is composed of the following elements:

- Look-up table (LUT) - performs logic operations
- Flip-Flop (FF) - register that stores the result of the LUT
- Wires - connects elements to one another
- Input/Output (I/O) pads - these physical ports get data in and out of the FPGA

A-1 Xilinx Artix-7 AC701 Evaluation board details

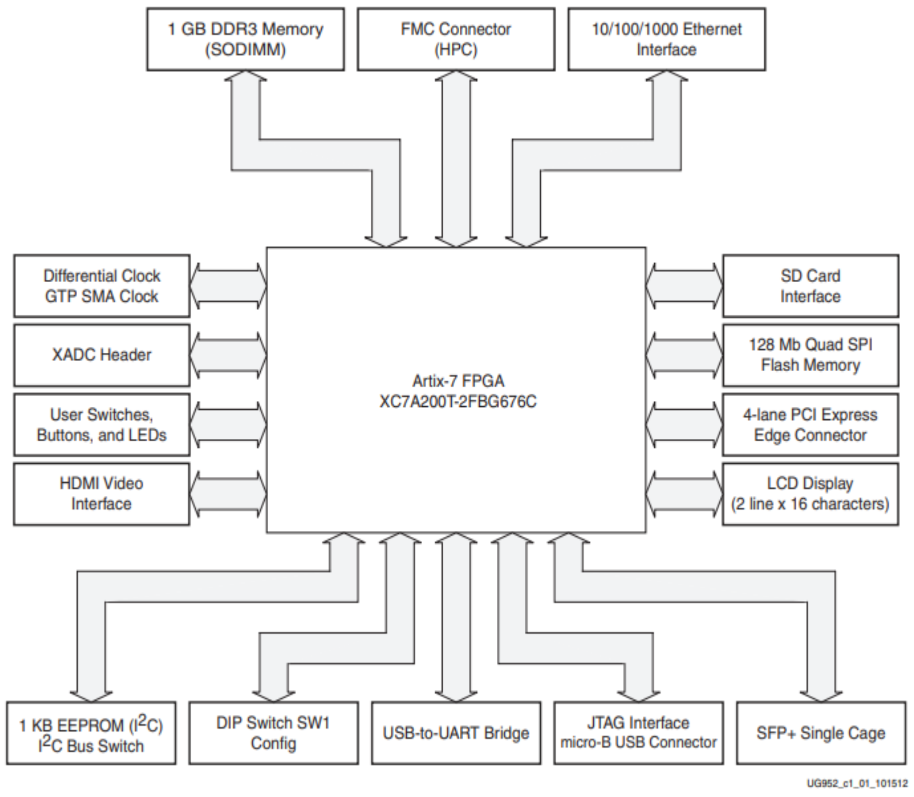

1 GB DDR3 Memory (SODIMM)
FMC Connector (HPC)
10/100/1000 Ethernet Interface
Differential Clock GTP SMA Clock
XADC Header
User Switches, Buttons, and LEDs
HDMI Video Interface
Artix-7 FPGA XC7A200T-2FBG676C
SD Card Interface
128 Mb Quad SPI Flash Memory
4-lane PCI Express Edge Connector
LCD Display (2 line x 16 characters)
1 KB EEPROM (I2C) I2C Bus Switch
DIP Switch SW1 Config
USB-to-UART Bridge
JTAG Interface micro-B USB Connector
SFP+ Single Cage
UG952_c1_01_101512

Figure A.1: Block diagram of the Xilinx AC701 evaluation board (Source: Guide to AC701 (2013))

The FPGA used in this work is the Xilinx Artix-7 FPGA AC701 Evaluation Kit which is a part of the Xilinx 7 series FPGAs. The block diagram of the AC701 and all its interfaces is shown in Figure A.1. AC701 board is used for the following reasons. First, the AC701 board contains a variety of communication interfaces, including a Secure Digital (SD) card connector and an HDMI Video interface, both of which are discussed in more detail in the following sections. Second, the AC701 board features 1 GB of DDR3 RAM and 13 MB of BRAM storage, which is ideal for storing intermediate data during processing. Third, the AC701 board allows high speed operations with a maximum clock frequency of 200 MHz. AC701 also supports clock domain variability, i.e board permits to use a wide range of user- defined clocks which is very useful while working with board interfaces.

A-2 SD Card interface

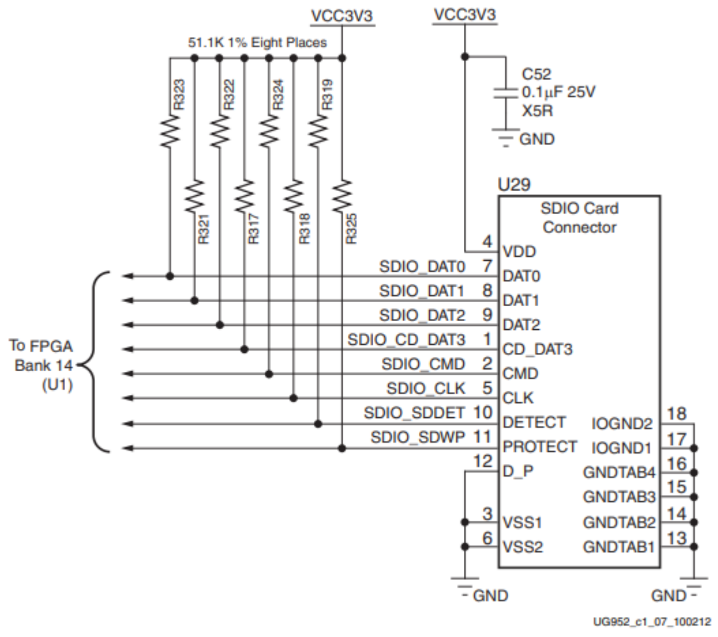


Figure A.2: SD card interface on the AC701 FPGA board. (Source: Guide to AC701 (2013))

Table 1 lists the SD card interface connections to the FPGA board. The SD card is a non-volatile, detachable memory storage device that is based on flash technology. Due to the qualities such as high throughput, low cost, high capacity, and low power consumption, SD cards are extensively used in multi-media applications. SD cards are available in a variety of memory storage capacities in the market. For typical operations, SD cards draw only 100 mA of current. All of these qualities combine to make the SD card an excellent medium for transferring image/video stream data into hardware architectures.

The SD card protocol supports 3 modes of operations: 1-bit, 4-bit and serial peripheral interface (SPI) modes. A Verilog code is written to implement the SPI mode of the SD card protocol and created a Verilog wrapper module to integrate it to the FPGA board and the other modules. Figure A.2 shows the block diagram and the signals present in the SD card interface. The AC701 board access non-volatile SDIO memory cards and peripherals using the secure digital input/output (SDIO) interface. The board supports 50 MHz high speed SD cards. The SDIO signals are connected to I/O bank 14, which has its VCCO set to 3.3 V.

Table 1: SDIO connections to the FPGA

FPGA pin (U1)	Schematic net name	I/O Standard count	U 29 SDIO connector	
			Pin number	Pin name
R 20	SDIO_SDWP	LVCMOS33	11	SDWP
P 24	SDIO_SDDET	LVCMOS33	10	SDDET
N23	SDIO_CMD	LVCMOS33	29	CMD
N 24	SDIO_CLK	LVCMOS33	5	CLK
P 23	SDIO_DAT2	LVCMOS33	9	DAT2
N 19	SDIO_DAT1	LVCMOS33	8	DAT1
P 19	SDIO_DAT0	LVCMOS33	7	DAT0
P 21	SDIO_CD_DAT3	LVCMOS33	1	CD_DAT3

(Source:Guide to AC701 (2013))

A-3 HDMI interface

High-definition multimedia interface (HDMI) is a digital video interface, designed to transmit uncompressed HD video data to a device that can display the data. HDMI's capabilities, compatibility and portability makes it the ideal protocol to use. The AC701 board uses Analog Devices ADV7511 chip to provide video output (U48). The ADV7511 is wired to support 1080P at 60 Hz, YCbCr 4:4:4 encoding using 24-bit input data mapping.

The AC701 board supports the following HDMI device interfaces:

- 24 data lines
- Independent VSYNC, HSYNC
- Single-ended input CLK
- Interrupt Out Pin to FPGA
- I2C
- SPDIF

Table 2 shows the list of connections between the codec and the FPGA. Table 3 lists the connections between the codec and the HDMI connector P2.

Table 2: FPGA to HDMI codec connections (ADV7511)

FPGA pin (U1)	Schematic net name	I/O Standard count	ADV7511 (U48)	
			Pin number	Pin name
AA24	HDM_I_R_D4	LVCMOS18	92	D4
Y25	HDMI_R_D5	LVCMOS18	91	D5
Y26	HDMI_R_D6	LVCMOS18	90	D6
V26	HDMI_R_D7	LVCMOS18	89	D7
W26	HDMI_R_D8	LVCMOS18	88	D8
W25	HDMI_R_D9	LVCMOS18	87	D9
W24	HDMI_R_D10	LVCMOS18	86	D10
U26	HDMI_R_D11	LVCMOS18	85	D11
U25	HDMI_R_D16	LVCMOS18	80	D16
V24	HDMI_R_D17	LVCMOS18	78	D17
U20	HDMI_R_D18	LVCMOS18	74	D18
W23	HDMI_R_D19	LVCMOS18	73	D19
W20	HDMI_R_D20	LVCMOS18	72	D20
U24	HDMI_R_D21	LVCMOS18	71	D21
Y20	HDMI_R_D22	LVCMOS18	70	D22
V23	HDMI_R_D23	LVCMOS18	69	D23
AA23	HDMI_R_D28	LVCMOS18	64	D28
AA25	HDMI_R_D29	LVCMOS18	63	D29
AB25	HDMI_R_D30	LVCMOS18	62	D30
AC24	HDMI_R_D31	LVCMOS18	61	D31
AB24	HDMI_R_D32	LVCMOS18	60	D32
Y22	HDMI_R_D33	LVCMOS18	59	D33
Y23	HDMI_R_D34	LVCMOS18	58	D34
V22	HDMI_R_D35	LVCMOS18	57	D35
AB26	HDMI_R_DE	LVCMOS18	97	DE
Y21	HDMI_R_SPDIF	LVCMOS18	10	SPDIF
V21	HDMI_R_CLK	LVCMOS18	79	CLK
AC26	HDMI_R_VSYNC	LVCMOS18	2	VSYNC
AA22	HDMI_R_HSYNC	LVCMOS18	98	HSYNC
W21	HDMI_INT	LVCMOS18	45	INT
T20	HDMI_SPDIF_OUT_LS	LVCMOS18	46	SPDIF_OUT

(Source:Guide to AC701 (2013))

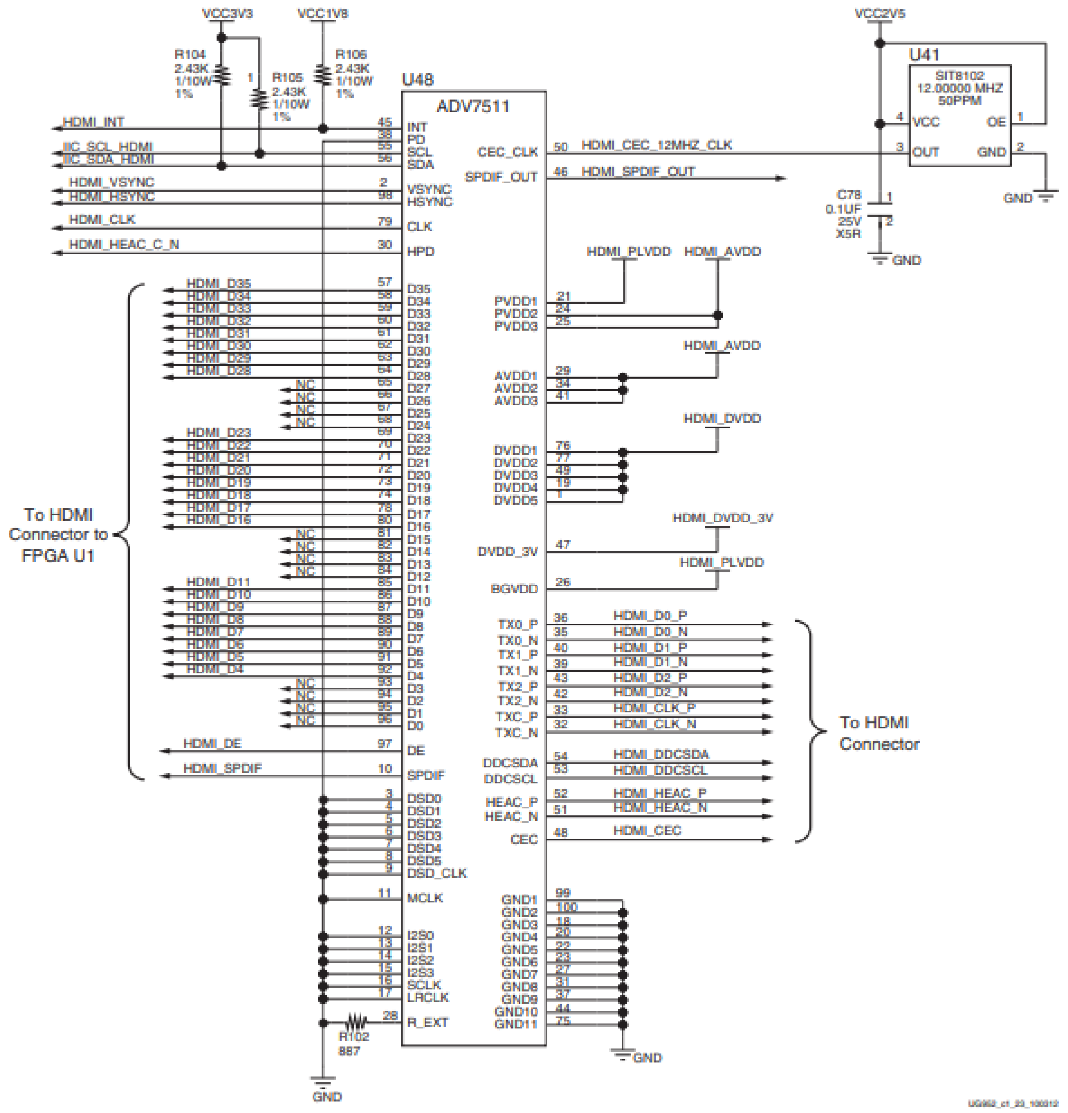


Figure A.3: HDMI codec circuit on the FPGA board. (Source: Guide to AC701 (2013))

A-4 System clock source

The AC701 board has a 2.5 V low voltage differential signalling (LVDS) 200 MHz oscillator. The clock is soldered onto the back side of the board and wired to the multi-region clock capable (MRCC) clock input present on the board. The 200 MHz signal pair is named SYSCLK_P and SYSCLK_N, which are connected to FPGA U1 pins R3 and P3 respectively. The AC701 board system clock has the following features.

Table 3: ADV7511 connections to HDMI connector

ADV7511 (U48)	Schematic net name	HDMI connector P2 Pin
36	HDMI_D0_P	7
35	HDMI_D0_N	9
40	HDMI_D1_P	4
39	HDMI_D1_N	6
43	HDMI_D2_P	1
42	HDMI_D2_N	3
33	HDMI_CLK_P	10
32	HDMI_CLK_N	12
54	HDMI_DDCSDA	16
53	HDMI_DDCSCL	15
52	HDMI_HEAC_P	14
51	HDMI_HEAC_N	19
48	HDMI_CRC	13

(Source: Guide to AC701 (2013))

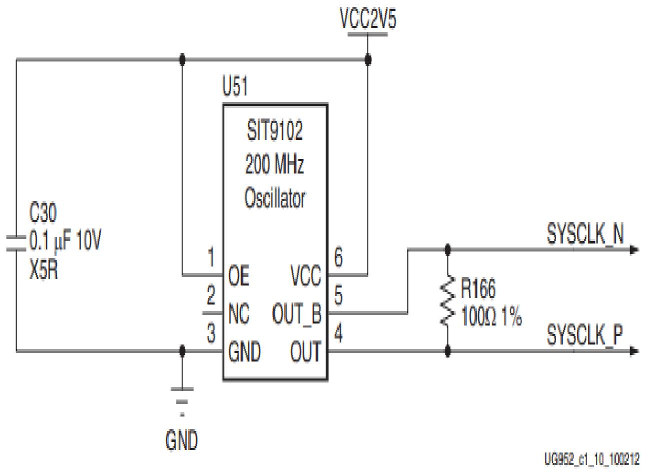


Figure A.4: System clock source present on the FPGA board. (Source: Guide to AC701 (2013))

- Oscillator: Si Time SiT9102AI-243N25E200.00000 (200 MHz)
- PPM (parts per million) frequency tolerance: 50 ppm
- Differential output

The block diagram of the system clock circuit is shown in Figure A.4. The FPGA board allows to use programmable low-jitter 3.3 V differential oscillator (U34) driving the FPGA MRCC inputs of bank 14. The board allows to connect to an external high-precision clock signal to be used with the FPGA bank 15 by connecting differential clock signals through the onboard 50 ohm surface mount assembly (SMA) connectors J31 (P) and J32 (N).

A-5 DSP48 block

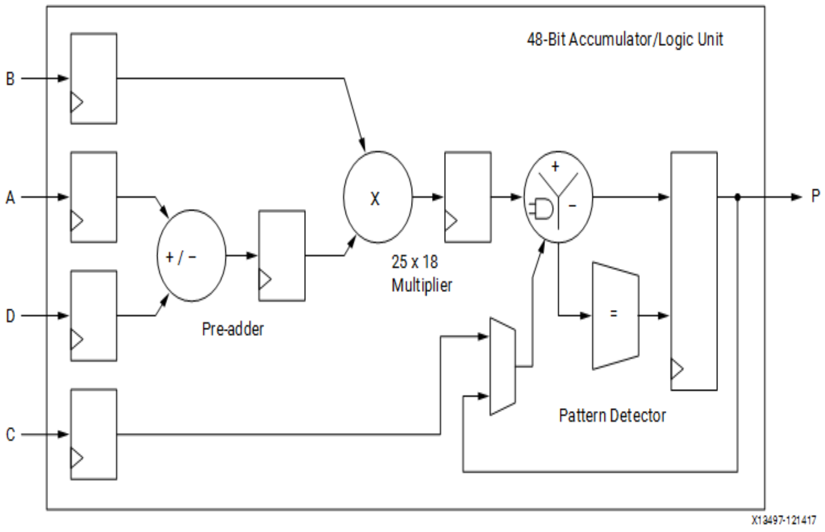


Figure A.5: Structure of a DSP48 block on the FPGA board. (Source: Guide to AC701 (2013))

The most complex computational block on the Xilinx FPGA is the DSP48 block, shown in Figure A.5. The DSP48 consists of an arithmetic logic unit (ALU) embedded into the FPGA fabric. There is a chain of three different computational blocks in the DSP48. This block chain is made of an add/subtract unit that is connected to a multiplier, which is then connected to a final add/subtract/accumulate engine. This chain allows a single DSP48 unit to implement functions of the form: $P = B \times (A + D) + C$ or $P+ = B \times (A + D)$. In this work, DSP slice is used to perform the function where A and B inputs are multiplied and the result is added to the C.

Bibliography

Abeydeera, M., **M. Karunaratne**, **G. Karunaratne**, **K. De Silva**, and **A. Pasqual** (2016). 4K Real-Time HEVC Decoder on an FPGA. *IEEE Transactions on Circuits and Systems for Video Technology*, **26**(1), 236–249.

Abramowski, A. and **G. Pastuszak**, A double-path intra prediction architecture for the hardware H.265/HEVC encoder. *In 17th International Symposium on Design and Diagnostics of Electronic Circuits Systems*. 2014, 27–32.

AMD Xilinx (2012). Xilinx 7 Series FPGAs: The Logical Advantage. [Online] Available: https://www.Xilinx WP405 Xilinx 7 Series FPGAs: The Logical Advantage, White Paper.

Amish, F. and **E.-B. Bourennane** (2016). Fully pipelined real time hardware solution for high efficiency video coding (HEVC) intra prediction. *Journal of Systems Architecture*, **64**, 133–147.

Amish, F. and **E.-B. Bourennane** (2019). An efficient hardware solution for 3D-HEVC intra-prediction. *Journal of Real-Time Image Processing*, **16**(5), 1559–1571.

Atapattu, S., **N. Liyanage**, **N. Menuka**, **I. Perera**, and **A. Pasqual**, Real time all intra HEVC HD encoder on FPGA. *In 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2016, 191–195.

Baldev, S., **P. K. Rathore**, **R. Peesapati**, and **K. K. Anumandla** (2021). A directional and scalable streaming deblocking filter hardware architecture for HEVC decoder. *Microprocessors and Microsystems*, **84**, 104029.

Baldev, S., **K. Shukla**, **S. Gogoi**, **P. K. Rathore**, and **R. Peesapati** (2018). Design and Implementation of Efficient Streaming Deblocking and SAO Filter for

HEVC Decoder. *IEEE Transactions on Consumer Electronics*, **64**(1), 127–135.

Bossen, F., **B. Bross**, **K. Suhring**, and **D. Flynn** (2012). HEVC complexity and implementation analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, **22**(12), 1685–1696.

Boyce, J., **D. Hong**, and **W. Jang**, Joint collaborative team on video coding (JCT-VC) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11. *In HEVC HM10 reference software (JCTVC-l1010), 12th meeting: Geneva, CH*. 2013, 14–23.

Chiang, P.-T., **Y.-C. Ting**, **H.-K. Chen**, **S.-Y. Jou**, **I.-W. Chen**, **H.-C. Fang**, and **T.-S. Chang** (2016). A QFHD 30-frames/s HEVC decoder design. *IEEE Transactions on Circuits and Systems for Video Technology*, **26**(4), 724–735.

Choi, Y. and **J. Joo** (2015). Exploration of practical HEVC/H. 265 sample adaptive offset encoding policies. *IEEE Signal Processing Letters*, **22**(4), 465–468.

Choudhury, R. and **P. Rangababu**, Design and Implementation of Mixed Parallel and Dataflow Architecture for Intra-prediction Hardware in HEVC Decoder. *In International Symposium on VLSI Design and Test*. Springer, 2017, 742–750.

Cisco Systems, Inc (2020). Cisco Annual Internet Report (2018–2023) White Paper, March, 2020. [Online] Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf. Accessed: Feb. 2022.

Ding, D., **S. Wang**, **Z. Liu**, and **Q. Yuan** (2019). Real-Time H. 265/HEVC Intra Encoding with a Configurable Architecture on FPGA Platform. *Chinese Journal of Electronics*, **28**(5), 1008–1017.

Diniz, C. M., **M. Shafique**, **F. V. Dalcin**, **S. Bampi**, and **J. Henkel**, A deblocking filter hardware architecture for the high efficiency video coding standard. *In 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, 1509–1514.

Fan, Y., **G. Tang**, and **X. Zeng** (2019). A Compact 32-Pixel TU-Oriented and SRAM-Free Intra Prediction VLSI Architecture for HEVC Decoder. *IEEE Access*, **7**, 149097–149104.

Fraunhofer (2015). HM, The reference software for HEVC. (HEVC Test Model). [Online] Available: https://hevc.hhi.fraunhofer.de/. Accessed: Feb. 2022, 1–5.

Fu, C.-M., **E. Alshina**, **A. Alshin**, **Y.-W. Huang**, **C.-Y. Chen**, **C.-Y. Tsai**, **C.-W. Hsu**, **S.-M. Lei**, **J.-H. Park**, and **W.-J. Han** (2012). Sample adaptive offset in the HEVC standard. *IEEE Transactions on Circuits and Systems for Video technology*, **22**(12), 1755–1764.

Fu, C.-M., **C.-Y. Chen**, **Y.-W. Huang**, and **S. Lei**, Sample adaptive offset for HEVC. *In 2011 IEEE 13th International Workshop on Multimedia Signal Processing*. IEEE, 2011, 1–5.

G. J. Sullivan (2021). Deployment Status of the HEVC Standard, ITU-T SD 16 WO3 - ISO/IEC JTC 1/SC 29/WG 11, Document JCTVC-AN0020-v1. [Online] Available: https://ultrahdforum.org/wp-content/uploads/UHD-Guidelines-V2.5-Fall2021.pdf. Accessed: Feb. 2022.

Grois, D., **D. Marpe**, **A. Mulayoff**, **B. Itzhaky**, and **O. Hadar**, Performance comparison of H. 265/MPEG-HEVC, VP9, and H. 264/MPRG-AVC Encoders. *In Picture Coding Symposium (PCS), 2013*. IEEE, 2013, 394–397.

Guide to AC701, G. S. (2013). Artix-7 FPGA AC701 Evaluation Kit (Vivado Design Suite.

Guide to Zed, G. S. (2012). Zed Board, (Zynq Evaluation and Development) Hardware User's Guide.

Haivision (2020). The Essential Guide to Video Encoding. [Online] Available: https://www.haivision.com/resources/white-paper/the-essential-guide-to-low-latency-video-streaming. Accessed: Feb. 2022.

Huang, C.-T., **M. Tikekar**, and **A. P. Chandrakasan** (2014). Memory-hierarchical and mode-adaptive HEVC intra prediction architecture for quad full HD video decoding. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **22**(7), 1515–1525.

Kalali, E., **Y. Adibelli**, and **I. Hamzaoglu** (2018). A low energy intra prediction hardware for high efficiency video coding. *Journal of Real-Time Image Processing*, **15**(2), 221–234.

Kim, I.-K., **J. Min**, **T. Lee**, **W.-J. Han**, and **J. Park** (2012). Block partitioning structure in the HEVC standard. *IEEE transactions on circuits and systems for video technology*, **22**(12), 1697–1706.

Kopperundevi, P., **M. S. Prakash**, and **S. R. Ahamed** (2022). A high throughput hardware architecture for deblocking filter in HEVC. *Signal Processing: Image Communication*, **100**, 116517.

Lainema, J., **F. Bossen**, **W.-J. Han**, **J. Min**, and **K. Ugur** (2012). Intra coding of the HEVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, **22**(12), 1792–1801.

Lainema, J. and **K. Ugur**, Angular intra prediction in high efficiency video coding (HEVC). *In 2011 IEEE 13th International Workshop on Multimedia Signal Processing*. IEEE, 2011, 1–5.

Li, F., **G. Shi**, and **F. Wu**, An efficient VLSI architecture for 4× 4 intra prediction in the High Efficiency Video Coding (HEVC) standard. *In Image Processing (ICIP), 2011 18th IEEE International Conference on Image Processing*. IEEE, 2011, 373–376.

Liu, L., **Y. Chen**, **C. Deng**, **S. Yin**, and **S. Wei** (2017). Implementation of in-loop filter for HEVC decoder on reconfigurable processor. *IET Image Processing*, **11**(9), 685–692.

Min, B., **Z. Xu**, and **R. C. C. Cheung** (2017). A Fully Pipelined Hardware Architecture for Intra Prediction of HEVC. *IEEE Transactions on Circuits and Systems for Video Technology*, **27**(12), 2702–2713.

NI CORP (2021). FPGA Fundamentals. [Online] Available: https://www.ni.com/en-in/innovations/white-papers/08/fpga-fundamentals.html. Accessed: Feb. 2022.

Norkin, A., **G. Bjontegaard**, **A. Fuldseth**, **M. Narroschke**, **M. Ikeda**, **K. Andersson**, **M. Zhou**, and **G. Van der Auwera** (2012). HEVC deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology*, **22**(12), 1746–1754.

Norkin, A., **C.-M. Fu**, **Y.-W. Huang**, and **S. Lei**, In-loop filters in hevc. *In High Efficiency Video Coding (HEVC)*. Springer, 2014, 171–208.

Ohm, J.-R. and **G. J. Sullivan** (2012). High efficiency video coding: the next frontier in video compression [standards in a nutshell]. *IEEE Signal Processing Magazine*, **30**(1), 152–158.

Ohm, J.-R., **G. J. Sullivan**, **H. Schwarz**, **T. K. Tan**, and **T. Wiegand** (2012). Comparison of the coding efficiency of video coding standards-including high efficiency video coding (HEVC). *IEEE Transactions on circuits and systems for video technology*, **22**(12), 1669–1684.

Onishi, T., **T. Sano**, **Y. Nishida**, **K. Yokohari**, **K. Nakamura**, **K. Nitta**, **K. Kawashima**, **J. Okamoto**, **N. Ono**, **A. Sagata**, *et al.* (2018). A Single-Chip 4K 60-fps 4: 2: 2 HEVC Video Encoder, LSI, Employing, Efficient, Motion Estimation and Mode Decision Framework With Scalability to 8K. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **26**(10), 1930–1938.

Ozcan, E., **Y. Adibelli**, and **I. Hamzaoglu** (2013). A high performance deblocking filter hardware for high efficiency video coding. *IEEE Transactions on Consumer Electronics*, **59**(3), 714–720.

Park, S., **J. Im**, and **K. Ryoo** (2016). Hardware Design of In-loop Filter for High Performance HEVC Encoder. *Journal of the Korea Institute of Information and Communication Engineering*, **20**(2), 335–342.

Pastuszak, G. and **A. Abramowski** (2016). Algorithm and Architecture Design of the H. 265/HEVC Intra Encoder. *IEEE Trans. Circuits Syst. Video Techn.*, **26**(1), 210–222.

Peesapati, R., **S. Das**, **S. Baldev**, and **S. R. Ahamed** (2017). Design of streaming deblocking filter for HEVC decoder. *IEEE Transactions on Consumer Electronics*, **63**(3), 1–9.

Rediess, F., **R. Conceição**, **B. Zatt**, **M. Porto**, and **L. Agostini**, Sample adaptive offset filter hardware design for HEVC encoder. *In 2014 IEEE Visual Communications and Image Processing Conference*. IEEE, 2014, 299–302.

Shen, S., **W. Shen**, **Y. Fan**, and **X. Zeng** (2013). A pipelined VLSI architecture for Sample Adaptive Offset (SAO) filter and deblocking filter of HEVC. *IEICE Electronics Express*, 10–20130272.

Shen, W., **Y. Fan**, **Y. Bai**, **L. Huang**, **Q. Shang**, **C. Liu**, and **X. Zeng** (2016). A combined deblocking filter and SAO hardware architecture for HEVC. *IEEE Transactions on Multimedia*, **18**(6), 1022–1033.

Shen, W., **Q. Shang**, **S. Shen**, **Y. Fan**, and **X. Zeng**, A high-throughput VLSI architecture for deblocking filter in HEVC.*In 2013 IEEE International Symposium on Circuits and Systems (ISCAS)/.* IEEE, 2013, 673–676.

Shukla, K., **B. Swamy**, and **P. Rangababu**, Area efficient dataflow hardware design of SAO filter for HEVC. *In 2017 international conference on innovations in electronics, signal processing and communication (IESC).* IEEE, 2017, 16–21.

Singhadia, A., **M. Minhazuddin**, **M. Mamillapalli**, and **I. Chakrabarti** (2021). A fast integrated deblocking filter and sample-adaptive-offset parameter estimation architecture for HEVC. *Microprocessors and Microsystems*, **85**, 104317.

Sjövall, P., **J. Virtanen**, **J. Vanne**, and **T. D. Hämäläinen**, High-level synthesis design flow for HEVC intra encoder on SoC-FPGA. *In 2015 Euromicro Conference on Digital System Design.* IEEE, 2015, 49–56.

Spin Digital (2020). HEVC Real-time Software Encoder for 8K Live Video Applications White Paper, December, 2020. [Online] Available: https://spin-digital.com/wp-content/uploads/2020/12/Whitepaper 8K-HEVC-Live-Encoder v1.1.1.pdf. Accessed: Feb. 2022.

Srinivasarao, B. K. N., **I. Chakrabarti**, and **M. N. Ahmad** (2015). High-speed low-power very-large-scale integration architecture for dual-standard deblocking filter. *IET Circuits, Devices & Systems*, **9**(5), 377–383.

Sullivan, G. J., **J. M. Boyce**, **Y. Chen**, **J.-R. Ohm**, **C. A. Segall**, and **A. Vetro** (2013). Standardized extensions of high efficiency video coding (HEVC). *IEEE Journal of selected topics in Signal Processing*, **7**(6), 1001–1016.

Sullivan, G. J. and **J.-R. Ohm** (2010). Recent developments in standardization of high efficiency video coding (HEVC). *Applications of Digital Image Processing XXXIII.* **7798**, SPIE, 239–245.

Sullivan, G. J., **J.-R. Ohm**, **W.-J. Han**, **T. Wiegand**, *et al.* (2012). Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology*, **22**(12), 1649–1668.

Sze, V., **M. Budagavi**, and **G. J. Sullivan**, High efficiency video coding (HEVC). volume 39. Springer, 2014, 40.

Telestream (2020). Advanced HEVC encoding considerations Ultilizing the full HEVC toolkit for greater efficiency and image quality. [Online] Available: http://www.telestream.net/pdfs/whitepapers/wp-HEVC.pdf. Accessed: Feb. 2022.

Tsai, S.-F., **C.-H. Tsai**, and **L.-G. Chen** (2014). *In 2017 international conference on innovations in electronics, signal processing and communication (IESC)*. IEEE, 2017, 16–21.

Ultra HD Forum (2021). Ultra HD Forum Guidelines. [Online] Available: https://ultrahdforum.org/wp-content/uploads/UHD-Guidelines-V2.5-Fall2021.pdf. Accessed: Feb. 2022.

Wang, Y., **X. Guo**, **Y. Lu**, **X. Fan**, and **D. Zhao**, Gpu-based optimization for sample adaptive offset in hevc. *In 2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, 829–833.

Wang, Y., **J. Ostermann**, and **Y.-Q. Zhang**, *Video processing and communications*, volume 1. Prentice hall Upper Saddle River, NJ, 2002.

Wiegand, T. and **G. J. Sullivan** (2007). The H. 264/AVC video coding standard [Standards in a Nutshell]. *IEEE Signal Processing Magazine*, **24**(2), 148–153.

Wiegand, T., **G. J. Sullivan**, **G. Bjontegaard**, and **A. Luthra** (2003). Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology*, **13**(7), 560–576.

Wien, M. (2015). High efficiency video coding. *Coding Tools and specification*, 133–160.

Wien, M., *High Efficiency Video Coding: Coding Tools and Specification*. Berlin, Germany:Springer-Verlag, Jan. 2015.

Yan, C., **Y. Zhang**, **F. Dai**, **X. Wang**, **L. Li**, and **Q. Dai** (2014). Parallel deblocking filter for HEVC on many-core processor. *Electronics Letters*, **50**(5), 367–368.

Zhang, Y. and **C. Lu** (2018). Efficient algorithm adaptations and fully parallel hardware architecture of H. 265/HEVC intra encoder. *IEEE Transactions on Circuits and Systems for Video Technology*, **29**(11), 3415–3429.

Zhao, L., **L. Zhang**, **S. Ma**, and **D. Zhao**, Fast mode decision algorithm for intra prediction in HEVC. *In 2011 Visual Communications and Image Processing (VCIP)*. IEEE, 2011, 1–4.

Zhou, M., **W. Gao**, **M. Jiang**, and **H. Yu** (2012). HEVC lossless coding and improvements. *IEEE Transactions on Circuits and Systems for Video Technology*, **22**(12), 1839–1843.

Zhu, J., **D. Zhou**, and **S. Goto** (2013*a*). A high performance HEVC de-blocking filter and SAO architecture for UHDTV decoder. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **96**(12), 2612–2622.

Zhu, J., **D. Zhou**, **G. He**, and **S. Goto**, A combined SAO and de-blocking filter architecture for HEVC video decoder. *In 2013 IEEE International Conference on Image Processing*. IEEE, 2013*b*, 1967–1971.

Publications Based on the Thesis

Journals:

1. **Lakshmi**, & Aparna, P. (2022). An efficient parallel-pipelined intra prediction architecture to support DCT/DST engine of HEVC encoder. *Journal of Real-Time Image Processing*, Springer (SCI and SCIE Indexed), Vol. 19, No. 3, pp. 539–550.
   https://doi.org/10.1007/s11554-022-01206-2
2. **Lakshmi**, & and Aparna P. (2020)., A Mixed Parallel and Pipelined Efficient Architecture for Intra Prediction Scheme in HEVC, *IETE Technical Review*, Taylor & Francis Online (SCI and SCIE Indexed), Vol. 39, No. 2, pp. 244–256.
   https://doi.org/10.1080/02564602.2020.1841686.
3. **Lakshmi**, Mohammad Asif Ansari, & Aparna P. (2022). Hardware Efficient Integrated In-loop Filter for HEVC Encoder, *IETE Technical Review*, Taylor & Francis. (Communicated).

Conferences:

1. **Lakshmi**, & Aparna P., Efficient Architectures for Planar and DC modes of Intra Prediction in HEVC, *In 7th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India*, pp. 148-153, IEEE, 2020.
   https://doi.org/10.1109/SPIN48934.2020.9071303.
2. Shwetha Shastri, **Lakshmi**, & Aparna P, Complexity Analysis of Hardware Architectures for Intra Prediction unit of High Efficiency Video Coding (HEVC). *In IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India*, pp. 1-6, IEEE, 2020.
   https://doi.org/10.1109/CONECCT50063.2020.9198553.

BIODATA

NAME : LAKSHMI

CONTACT DETAILS

Address : G3, Poojan Enclave, Gandhinagar,

Mangalore, India-575003.

Email ✉ : lakshmipoola@gmail.com

Mobile No. 📱 : +91 9845510232.

EDUCATIONAL QUALIFICATIONS

Doctor of Philosophy (Ph.D)

National Institute of Technology Karnataka, Surathkal 2017–Till date

Master of Technology (M.Tech)

NMAMIT, Nitte. 2012–2014

Specialization: VLSI Design and Embedded Systems.

Bachelor of Engineering (B.E)

UVCE, Bangalore. 1998–2002

Branch: Electronics and Communication Engineering.

RESEARCH INTERESTS

VLSI design, algorithms, and FPGA implementation of VLSI architectures for multimedia signal processing.

PUBLICATIONS

Number of Journals : 2
Number of Conferences : 4

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

DECLARATION

I hereby ***declare*** that the Research Thesis entitled **AREA EFFICIENT HARDWARE ARCHITECTURES OF INTRA PREDICTION AND SAMPLE ADAPTIVE OFFSET FILTER FOR HEVC ENCODER** which is being submitted to the ***National Institute of Technology Karnataka, Surathkal*** in partial fulfilment of the requirements for the award of the Degree of ***Doctor of Philosophy*** in **Department of Electronics and Communication Engineering** is a ***bonafide report of the research work carried out by me***. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

[illegible]
13/01/23
LAKSHMI
Reg. No. 177034/177EC006
Department of Electronics and
Communication Engineering,
NITK, Surathkal.

Place: NITK-Surathkal.
Date: 13.01.2023

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

CERTIFICATE

This is to certify that the Research Thesis entitled **AREA EFFICIENT HARDWARE ARCHITECTURES OF INTRA PREDICTION AND SAMPLE ADAPTIVE OFFSET FILTER FOR HEVC ENCODER** submitted by **LAKSHMI** (Register Number: 177EC006/177034) as the record of the research work carried out by her, is accepted as the *Research Thesis submission* in partial fulfilment of the requirements for the award of degree of **Doctor of Philosophy**.

[illegible] 13/1/2023

Dr. Aparna P.
Research Supervisor
Assistant Professor
Dept. of E&C Engg.
NITK Surathkal - 575025.

[illegible]
13-1-2.23

Chairman-DRPC
Dept. of E&C Engg.
NITK Surathkal - 575025.
(Signature with Date and Seal)

HEAD
[illegible] ant
[illegible] hkal
मंगलूर / MANGALORE - 575 025