LOW ENERGY HEVC AND VVC VIDEO COMPRESSION HARDWARE

by

Hasan Azgın

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Sabancı University

August 2019

LOW ENERGY HEVC AND VVC VIDEO COMPRESSION HARDWARE

APPROVED BY:

Assoc. Prof. Dr İlker Hamzaoğlu
(Thesis Supervisor)

Assoc. Prof. Dr. Hüsnü Yenigün

Asst. Prof. Dr. Hüseyin Özkan

Asst. Prof. Dr. Yakup Genç

Asst. Prof. Dr. Alp Arslan Bayrakçi

DATE OF APPROVAL: 19/07/2019

© Hasan Azgın 2019
All Rights Reserved

To my Family

ACKNOWLEDGEMENT

I would like to thank my thesis supervisor, Dr. İlker Hamzaoğlu for all his guidance, support, and patience throughout my PhD study. I am grateful for not only his detailed reviews, academical guidance, but also his life lessons and talks on the life. I particularly want to thank him for his belief in me during my study. It has been a great honor for me to work under his guidance.

I would like to thank to the members of System-on-Ship Design and Testing Lab; Ercan Kalalı and Ahmet Can Mert for their great friendship and their collaboration during my studies.

I would like to give my special thanks to my mother, my father and my brother. They have always believed in me and supported me in the good times and the bad times. This thesis is dedicated with love to them.

Finally, I would like to thank Sabanci University and Scientific and Technological Research Council of Turkey (TUBITAK) for supporting me throughout my graduate education. This thesis was supported by TUBITAK under the contracts 115E290 and 118E134.

ABSTRACT

LOW ENERGY HEVC AND VVC VIDEO COMPRESSION HARDWARE

Hasan Azgın
Electronics, PhD Dissertation, 2019

Thesis Supervisor: Assoc. Prof. İlker Hamzaoğlu

Keywords: HEVC, VVC, Intra Prediction, Fractional Interpolation, Approximate Computing, Hardware Implementation, FPGA, Low Energy, DSP

Video compression standards compress a digital video by reducing and removing redundancy in the digital video using computationally complex algorithms. As spatial and temporal resolutions of videos increase, compression efficiencies of video compression algorithms are also increasing. However, increased compression efficiency comes with increased computational complexity. Therefore, it is necessary to reduce computational complexities of video compression algorithms without reducing their visual quality in order to reduce area and energy consumption of their hardware implementations.

In this thesis, we propose a novel technique for reducing amount of computations performed by HEVC intra prediction algorithm. We designed low energy, reconfigurable HEVC intra prediction hardware using the proposed technique. We also designed a low energy FPGA implementation of HEVC intra prediction algorithm using the proposed technique and DSP blocks. We propose a reconfigurable VVC intra prediction hardware architecture. We also propose an efficient VVC intra prediction hardware architecture using DSP blocks. We designed low energy VVC fractional interpolation hardware. We propose a novel approximate absolute difference technique. We designed low energy approximate absolute difference hardware using the proposed technique. We propose a novel approximate constant

multiplication technique. We designed approximate constant multiplication hardware using the proposed technique.

We quantified computation reductions achieved by the proposed techniques and video quality loss caused by the proposed approximation techniques. The proposed approximate absolute difference technique and approximate constant multiplication technique cause very small PSNR loss. The other proposed techniques cause no PSNR loss. We implemented the proposed hardware architectures in Verilog HDL. We mapped the Verilog RTL codes to Xilinx Virtex 6 or Xilinx Virtex 7 FPGAs and estimated their power consumptions using Xilinx XPower Analyzer tool. The proposed techniques significantly reduced power and energy consumptions of these FPGA implementations.

ÖZET

DÜŞÜK ENERJİLİ HEVC VE VVC VIDEO SIKIŞTIRMA DONANIMLARI

Hasan Azgın
Elektronik Müh., Doktora Tezi, 2019

Tez Danışmanı: Doç. Dr. İlker Hamzaoğlu

Anahtar Kelimeler: HEVC, VVC, Çerçeve İçi Öngörü, Kesirli Aradeğerleme, Yaklaşık Hesaplama, Donanım Gerçekleme, FPGA, Düşük Enerji, DSP

Video sıkıştırma standartları, bir sayısal videonun içindeki gereksiz bilgileri, yüksek hesaplama karmaşıklığına sahip algoritmalar yardımıyla, azaltarak veya kaldırarak videoyu sıkıştırır. Videoların zamansal ve uzaysal çözünürlüğü arttıkça, video sıkıştırma algoritmalarının sıkıştırma etkinliği de artmaktadır. Ancak bu artan sıkıştırma etkinliği, yüksek hesaplama karmaşıklığını da beraberinde getirmektedir. Bu yüzden, video sıkıştırma algoritmalarının donanımlarının alanını ve harcadıkları enerji miktarını azaltmak için, bu algoritmaların hesaplama karmaşıklığını, görsel kaliteyi düşürmeden azaltmak gereklidir.

Bu tezde, HEVC çerçeve içi öngörü algoritmasının hesaplama miktarını azaltmak için orijinal bir teknik önerilmektedir. Önerilen teknik kullanılarak, düşük enerjili, yeniden ayarlanabilir HEVC çerçeve içi öngörü donanımı tasarlanmıştır. Önerilen teknik ve DSP blokları kullanılarak, düşük enerjili bir HEVC çerçeve içi öngörü FPGA gerçeklemesi tasarlanmıştır. Yeniden ayarlanabilir VVC çerçeve içi öngörü mimarisi önerilmektedir. DSP bloklarının kullanıldığı, etkin bir VVC çerçeve içi öngörü mimarisi önerilmektedir. Düşük enerjili VVC kesikli aradeğerleme donanımı tasarlanmıştır. Orijinal bir yaklaşık mutlak fark hesaplama tekniği önerilmektedir. Önerilen teknik kullanılarak düşük enerjili yaklaşık mutlak değer hesaplama donanımları tasarlanmıştır. Orijinal bir yaklaşık sabit çarpma tekniği önerilmektedir. Önerilen teknik kullanılarak, yaklaşık sabit çarpma donanımı tasarlanmıştır.

Önerilen tekniklerin sağladığı hesaplama azaltmaları ve yaklaşık tekniklerin neden olduğu video kalitesi kayıpları ölçüldü. Önerilen yaklaşık mutlak değer tekniği ve yaklaşık sabit çarpma tekniği çok düşük PSNR kaybına neden oldu. Önerilen diğer teknikler ise PSNR kaybına neden olmadı. Önerilen donanım mimarileri Verilog donanım tasarlama dili ile gerçeklendi. Verilog RTL kodları Xilinx Virtex 6 veya Xilinx Virtex 7 FPGA'larına sentezlendi ve güç tüketimleri Xilinx XPower Analyzer aracı ile tahmin edildi. Önerilen teknikler, bu FPGA gerçeklemelerinin güç ve enerji tüketimlerini önemli ölçüde azalttı.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BRAM	Block RAM
CABAC	Context Adaptive Binary Arithmetic Coding
CU	Coding Unit
DBF	Deblocking Filter
DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
FPGA	Field Programmable Gate Array
HEVC	High efficiency Video Coding
HM	HEVC Test Model
IDCT	Inverse Discrete Cosine Transform
JEM	Joint Exploration Model
PSNR	Peak Signal to Noise Ratio
PU	Prediction Unit
QP	Quantization Parameter
TU	Transform Unit
VCD	Value Change Dump
VVC	Versatile Video Coding

CHAPTER I

INTRODUCTION

Temporal and spatial video resolutions are increasing. This is expected to continue in the future as well. To store or transmit this large amount of video data, video compression standards with high compression efficiency are needed. Joint Collaborative Team on Video Coding (JCT-VC) developed a video compression standard called High Efficiency Video Coding (HEVC) [1, 2, 3]. HEVC provides 50% better coding efficiency than the previous video compression standard, H.264. HEVC uses computationally more complex algorithms to provide better compression efficiency. Joint Video Experts Team (JVET) is developing a new video compression standard called Versatile Video Coding (VVC) [4], which is expected to be finalized in 2020. JVET provided a software model for the current version of VVC. Current version of VVC provides better compression efficiency than HEVC using computationally more complex algorithms.

Video compression standards compress a video by removing redundancies in the video such as spatial, temporal and statistical redundancies. There is spatial correlation between neighboring pixels in a video frame. Intra prediction and mode decision algorithms removes spatial redundancy by determining the correlation between neighboring blocks of pixels in a frame and encoding this correlation instead of pixel values. There is temporal correlation between neighboring frames of a video. Inter prediction and mode decision algorithms removes temporal redundancy by determining the correlation between blocks of pixels in neighboring frames and encoding this correlation instead of pixel values. There is statistical redundancy between the data that will be encoded. Entropy coding algorithms such as Huffman variable length coding algorithm remove statistical redundancy by representing the more frequently occurring data with small number of bits and less frequently occurring data with large number of bits.

Approximate computing is a promising solution to increased computational complexity of video compression algorithms [5]-[9]. Approximate computing allows designing faster, lower area and lower power consuming hardware than the exact optimized hardware by trading off speed, area and power consumption with quality. Therefore, it can be used in error tolerant applications such as video compression.

1.1 HEVC Video Compression Standard

HEVC is the current state-of-the-art video compression standard developed by Collaborative Team on Video Coding (JCT-VC). HEVC video compression standard consists of several video compression algorithms such as intra prediction, motion estimation, transform, quantization and entropy coder. The top-level block diagram of HEVC encoder and HEVC decoder are shown in Figure 1.1 and Figure 1.2, respectively.

HEVC encoder has a forward path and a reconstruction path. The forward path generates bitstream. A frame is divided into 8x8, 16x16, 32x32 or 64x64 coding units (CU). A CU can be divided into prediction units (PU). PU sizes are from 4x4 up to 64x64. PU size can be the same as or less than the size of current CU. Motion estimation determines the best inter prediction for the current CU. Intra prediction determines the best intra prediction for the current CU. Mode decision determines the best prediction among them and PU size in terms of video quality and bit rate. Residue, difference between the current CU and the best prediction, is encoded using transform, quantization and entropy coder algorithms to generate bitstream. Since HEVC decoder does not have access to the original frame, reconstruction path in the encoder is used to prevent mismatch between encoder and decoder. By using reconstruction path, identical reference frames are used in both encoder and decoder.

Reconstruction path begins with inverse quantization and inverse transform to generate the reconstructed residue. Since quantization is a lossy process, inverse quantized and inverse transformed coefficients are not identical to the original residue. Reconstructed frame is generated by adding the reconstructed residue to the predicted pixels. Blocking artifacts are reduced by using deblocking filter (DBF) algorithm.

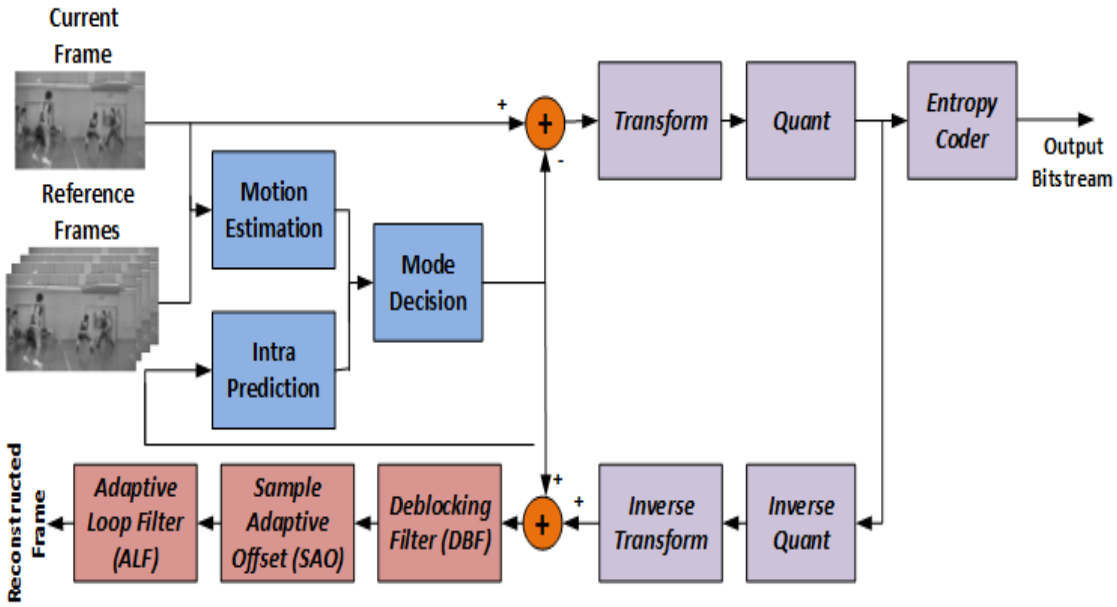

Figure 1.1 HEVC Encoder Block Diagram

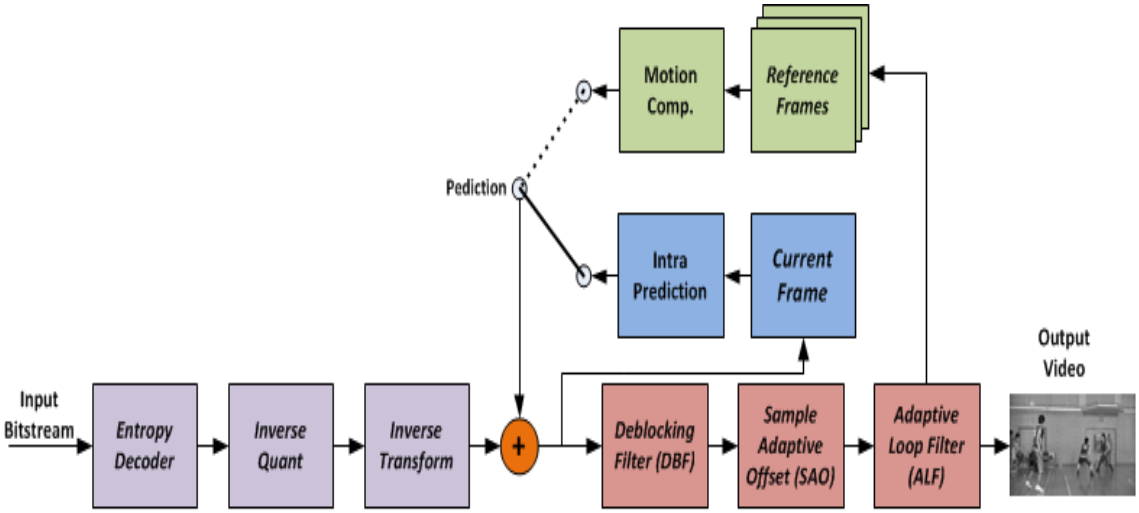

Figure 1.2 HEVC Decoder Block Diagram

HEVC intra prediction algorithm predicts the pixels of a block from the pixels of its already coded and reconstructed neighboring blocks in the same frame. For the luminance component of a frame, intra PU size can be from 4x4 up to 32x32 and number of intra prediction modes for a PU can be up to 35 [1, 2]. There are 33 angular prediction modes, DC and planar prediction modes. In angular prediction modes, predicted pixels are generated by weighted average of two neighboring pixels.

HEVC inter prediction algorithm predicts the pixels of a block in the current frame from the pixels of already coded and reconstructed blocks in the neighboring frames. Inter PU size can be from 4x8 and 8x4 up to 64x64. HEVC inter prediction algorithm, first, performs integer pixel motion estimation for a PU. Then, it performs fractional motion estimation for the same PU. It uses three different 8-tap FIR filters for generating half pixels and quarter pixels [1, 2].

HEVC uses discrete cosine transform (DCT) for transform unit (TU) sizes of square shapes from 4x4 up to 32x32. HEVC also uses discrete sine transform (DST) for 4x4 intra prediction case [1, 2]. Inverse discrete cosine transform (IDCT) and inverse discrete sine transform (IDST) are used in the reconstruction path of encoder and in the decoder.

HEVC entropy coder uses context adaptive binary arithmetic coding (CABAC) to generate output bitstream.

HEVC uses deblocking filter algorithm to reduce blocking artifacts on the edges of PUs.

1.2 VVC Video Compression Standard

JVET is currently developing a new video compression standard called Versatile Video Coding (VVC) [4]. VVC is not finalized yet. However, a software model implementing its current version is provided. The current version of VVC standard has much better coding efficiency than HEVC at the expense of much higher computational complexity [4]. VVC has a similar top-level block diagram to HEVC.

VVC intra prediction algorithm is similar to HEVC intra prediction algorithm. However, in VVC, number of angular intra prediction modes is increased to 65. In addition, VVC uses 4-tap cubic and 4-tap gaussian filters for angular intra prediction modes [12, 13].

VVC inter prediction algorithm performs the same two-stage search as HEVC. However, VVC performs fractional motion estimation at one sixteenth motion vector accuracy. It also has an improved motion vector prediction process [1, 2, 13].

VVC uses integer based DCT same as HEVC. However, VVC uses an Adaptive Multiple Transform (AMT) scheme which uses DCT-II, DCT-V, DCT-VIII, DST-I and DST-VII based on prediction type. In addition, VVC TU sizes can be from 4x4 up to 64x64 [13]-[16].

VVC entropy coder uses CABAC algorithm similar to HEVC entropy coder with several enhancements. VVC DBF algorithm is the same as HEVC DBF algorithm [1, 2, 13].

1.3 Thesis Contributions

We propose a novel technique for reducing amount of computations performed by HEVC intra prediction algorithm and, therefore, reducing energy consumption of HEVC intra prediction hardware. The proposed technique significantly reduced the amount of computations by reorganizing HEVC intra prediction equations. The proposed technique does not affect PSNR and bit rate. A low energy HEVC intra angular prediction hardware using the proposed technique is designed and implemented. The proposed technique significantly reduced energy consumption of the HEVC intra prediction hardware [18].

Since full-custom DSP blocks in Xilinx FPGAs perform constant multiplications faster and with less energy than adders and shifters, we propose an efficient FPGA implementation of HEVC intra prediction for angular prediction modes using the proposed computation and energy reduction technique and DSP blocks in FPGA. In the proposed FPGA implementation, one HEVC intra angular prediction equation is implemented using one DSP block instead of using two DSP blocks and two adders [19].

We propose two VVC reconfigurable intra prediction hardware. They are the first VVC intra prediction hardware in the literature. The first hardware implements multiplications with constants using adders and shifters instead of using multipliers. Therefore, it can be used in ASIC implementations of VVC encoders. The second hardware implements multiplications with constants using DSP blocks in FPGA instead of using adders and shifters. Therefore, it can be used in FPGA implementations of VVC encoders [20].

We propose an efficient FPGA implementation of VVC intra prediction for angular prediction modes. In the proposed FPGA implementation, intra angular prediction equations are manipulated in such a way that one intra angular prediction equation is implemented using two DSP blocks and two adders [21].

We propose a reconfigurable VVC fractional interpolation hardware for motion compensation. The proposed hardware has a reconfigurable datapath which can be configured to implement any of the 15 different 8-tap FIR filters used for fractional interpolation. Since the proposed hardware is used for motion compensation in VVC encoder and decoder, only one fractional pixel per integer pixel is interpolated [22].

We propose four novel approximate absolute difference hardware using special approximation techniques [23]. We propose a novel approximate constant multiplication technique. The proposed approximate constant multiplication technique decreases complexity of constant multiplication by converting it to a multiplication with a smaller constant, concatenation and constant shift operation. The proposed approximation techniques reduce area and power consumption of hardware implementations with negligible video quality loss.

1.4 Thesis Organization

The rest of the thesis is organized as follows.

Chapter II, first, explains HEVC intra prediction algorithm. It describes the proposed technique for reducing amount of computations performed by HEVC intra prediction. The proposed HEVC intra prediction hardware is explained and its implementation results are given. Then, the proposed FPGA implementation of HEVC intra prediction using the proposed technique and DSP blocks is explained. The implementation results are given. Finally, comparison of the proposed hardware with the ones proposed in literature is presented.

Chapter III, first, explains VVC intra prediction algorithm. The proposed reconfigurable VVC intra prediction hardware implementations are explained and their implementation results are given. Then, the proposed FPGA implementation of VVC intra prediction using DSP blocks is explained. The implementation results are given. Finally, comparison of the proposed hardware with the ones proposed in literature is presented.

Chapter IV, first, explains VVC fractional interpolation algorithm. Then, the proposed VVC fractional interpolation hardware and its reconfigurable datapath are explained. Finally, implementation results are given, and literature comparison is presented.

Chapter V, first, explains approximate computing. Then, the proposed novel approximate absolute difference technique is explained. The proposed four different approximate absolute difference hardware are presented. Their implementation results are given. They are compared with approximate absolute difference hardware implementations using the proposed approximate adders in literature. Then, the proposed novel approximate constant multiplication technique is explained. HEVC 2D transform and VVC 2D transform hardware implementations using the proposed approximate

constant multiplier are presented. Their rate-distortion performances and hardware implementation results are given.

Chapter VI presents conclusions and future works.

CHAPTER II

HEVC INTRA PREDICTION HARDWARE

2.1 HEVC Intra Prediction Algorithm

HEVC intra prediction algorithm predicts the pixels in prediction units (PU) of a coding unit (CU) using the pixels in the available neighboring PUs [1]. For the luminance component of a frame, 4x4, 8x8, 16x16 and 32x32 PU sizes are available. As shown in Figure 2.1, there are 33 angular prediction modes (Mode) corresponding to different prediction angles (Angle) for each PU size. In addition, there are DC and planar prediction modes for each PU size. An 8x8 PU, four 4x4 PUs in it, and their neighboring pixels are shown in Figure 2.2.

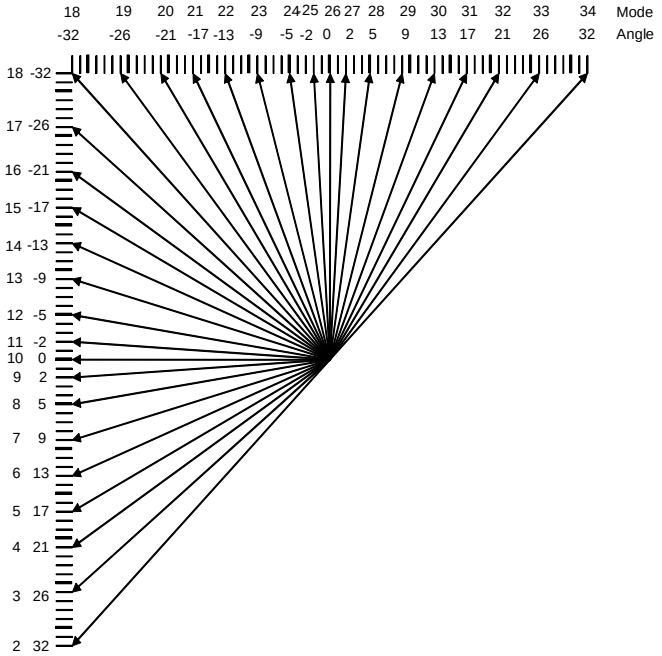

Figure 2.1 HEVC Intra Prediction Mode Directions

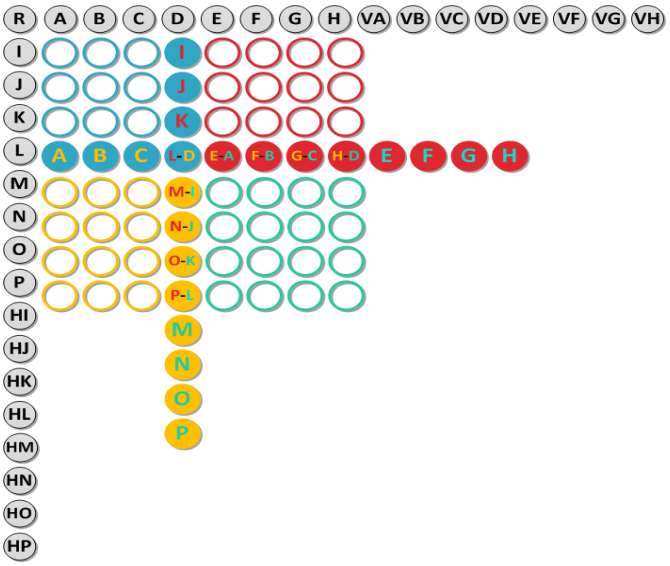

Figure 2.2 Neighboring Pixels of 4x4 and 8x8 PUs

In HEVC intra prediction algorithm, first, reference main array is determined. The pixels in the reference main array are used in the intra prediction equations. If the prediction mode is equal to or greater than 18, reference main array is selected from above neighboring pixels. However, first four pixels of this array are reserved to left neighboring pixels, and if prediction angle is less than zero, these pixels are assigned to the array. If the prediction mode is less than 18, reference main array is selected from left neighboring pixels. However, first four pixels of this array are reserved to above neighboring pixels, and if prediction angle is less than zero, these pixels are assigned to the array.

After the reference main array is determined, ildx which is used to determine positions of the pixels in this array that will be used in the intra prediction equations and iFact which is used to determine coefficients of these pixels are calculated as shown in equations (2.1a) and (2.1b), respectively. If iFact is equal to 0, neighboring pixels are copied directly to predicted pixels. Otherwise, predicted pixels are calculated as shown in equation (2.2).

$$iIdx = \big((y+1) * Angle\big) \gg 5 \tag{2.1a}$$

$$iFact = \big((y+1) * Angle\big) \,\&\, 31 \tag{2.1b}$$

$$pred[x,y] = ((32 - iFact) * refMain[x + iIdx + 1] + iFact * refMain[x + iIdx + 2] + 16) \gg 5 \tag{2.2}$$

All the intra prediction equations can be obtained from equation (2.2). As an example, reference main array and prediction equations for the 8x8 intra prediction mode 6 with prediction angle 13 are shown in equations (2.3a) and (2.3b), respectively. The neighboring pixels used in these equations can be seen in Figure 2.2.

$$x = 0\ to\ (PU_{size} - 1), y = 0\ to\ (PU_{size} - 1)$$

$$refMain = [0,0,0,0,0,0,0,0,R,A,B,C,D,E,F,G,H,VA,VB,VC,VD,VE,VF,VG,VH] \quad (2.3a)$$

$$\begin{gathered} pred[0,0] = pred[1,0] = [19 * A + 13 * B + 16] >> 5 \\ pred[2,0] = pred[3,0] = [19 * B + 13 * C + 16] >> 5 \\ pred[4,0] = pred[5,0] = pred[6,0] = [19 * C + 13 * D + 16] >> 5 \\ pred[7,0] = [19 * D + 13 * E + 16] >> 5 \end{gathered} \quad (2.3b)$$

$$\begin{gathered} pred[0,1] = pred[1,1] = [6 * B + 26 * C + 16] >> 5 \\ pred[2,1] = pred[3,1] = [6 * C + 26 * D + 16] >> 5 \\ pred[4,1] = pred[5,1] = pred[6,1] = [6 * D + 26 * E + 16] >> 5 \\ pred[7,1] = [6 * E + 26 * F + 16] >> 5 \end{gathered}$$

$$\begin{gathered} pred[0,2] = pred[1,2] = [25 * C + 7 * D + 16] >> 5 \\ pred[2,2] = pred[3,2] = [25 * D + 7 * E + 16] >> 5 \\ pred[4,2] = pred[5,2] = pred[6,2] = [25 * E + 7 * F + 16] >> 5 \\ pred[7,2] = [25 * F + 7 * G + 16] >> 5 \end{gathered}$$

$$\begin{gathered} pred[0,3] = pred[1,3] = [12 * D + 20 * E + 16] >> 5 \\ pred[2,3] = pred[3,3] = [12 * E + 20 * F + 16] >> 5 \\ pred[4,3] = pred[5,3] = pred[6,3] = [12 * F + 20 * G + 16] >> 5 \\ pred[7,3] = [12 * G + 20 * H + 16] >> 5 \end{gathered}$$

$$\begin{gathered} pred[0,4] = pred[1,4] = [31 * E + 1 * F + 16] >> 5 \\ pred[2,4] = pred[3,4] = [31 * F + 1 * G + 16] >> 5 \\ pred[4,4] = pred[5,4] = pred[6,4] = [31 * G + 1 * H + 16] >> 5 \\ pred[7,4] = [31 * H + 1 * I + 16] >> 5 \end{gathered}$$

$$\begin{gathered} pred[0,5] = pred[1,5] = [18 * F + 14 * G + 16] >> 5 \\ pred[2,5] = pred[3,5] = [18 * G + 14 * H + 16] >> 5 \\ pred[4,5] = pred[5,5] = pred[6,5] = [18 * H + 14 * VA + 16] >> 5 \\ pred[7,5] = [18 * VA + 14 * VB + 16] >> 5 \end{gathered}$$

$$\begin{gathered} pred[0,6] = pred[1,6] = [5 * G + 27 * H + 16] >> 5 \\ pred[2,6] = pred[3,6] = [5 * H + 27 * VA + 16] >> 5 \\ pred[4,6] = pred[5,6] = pred[6,6] = [5 * VA + 27 * VB + 16] >> 5 \\ pred[7,6] = [5 * VB + 27 * VC + 16] >> 5 \end{gathered}$$

$$\begin{gathered} pred[0,7] = pred[1,7] = [24 * H + 8 * VA + 16] >> 5 \\ pred[2,7] = pred[3,7] = [24 * VA + 8 * VB + 16] >> 5 \\ pred[4,7] = pred[5,7] = pred[6,7] = [24 * VB + 8 * VC + 16] >> 5 \\ pred[7,7] = [24 * VC + 8 * VD + 16] >> 5 \end{gathered}$$

2.2 A Computation and Energy Reduction Technique for HEVC Intra Prediction

In this thesis, a novel technique is proposed for reducing amount of computations performed by HEVC intra prediction algorithm and, therefore, reducing energy consumption of HEVC intra prediction hardware. The proposed technique reorganizes the HEVC intra prediction equations by utilizing the fact that the sum of the coefficients used in each HEVC angular intra prediction equation is 32. The reorganized intra prediction equations require less number of addition and shift operations than the original ones. This reduces the amount of computations performed by 4x4, 8x8, 16x16 and 32x32 luminance angular prediction modes. It does not affect the PSNR and bit rate.

In this thesis, a low energy HEVC intra prediction hardware for angular prediction modes of all PU sizes (4x4, 8x8, 16x16 and 32x32) is also designed and implemented using Verilog HDL. The Verilog RTL code is mapped to an FPGA implemented in 40 nm CMOS technology. The FPGA implementation is verified to work correctly on an FPGA board. The FPGA implementation can work at 166 MHz, and it can process 40 full HD (1920 x 1080) video frames per second. The proposed HEVC intra prediction hardware implementing the reorganized HEVC intra prediction equations has up to 24.63% less energy consumption than an HEVC intra prediction hardware implementing the original HEVC intra prediction equations.

Several HEVC intra prediction hardware implementations are proposed in the literature [24]-[33]. Some of them have higher performance than the proposed HEVC intra prediction hardware at the expense of much larger hardware area. The area of the proposed hardware is much smaller than the ones proposed in [24]-[32]. Some of these HEVC intra prediction hardware use separate hardware for each PU size. Some of them use many parallel intra prediction datapaths. Some of them use multipliers instead of adders and shifters for implementing multiplication with constants.

Power consumptions of the hardware implementations proposed in [24]-[31] are not reported. The proposed hardware consumes less power than the one proposed in [32]. The proposed HEVC intra prediction hardware implementation performs intra prediction for all PU sizes. Since the HEVC intra prediction hardware implementation proposed in [33] performs intra prediction only for 4x4 and 8x8 PU sizes, it has smaller area and consumes less power than the proposed one.

2.2.1 Proposed Computation and Energy Reduction Technique

In this thesis, data reuse technique is first used for reducing amount of computations performed by HEVC intra prediction algorithm. In HEVC, intra 4x4, 8x8, 16x16 and 32x32 luminance angular prediction modes have identical equations. There are identical equations between luminance angular prediction modes of different PU sizes as well. Data reuse technique calculates the common prediction equations for all 4x4, 8x8, 16x16 and 32x32 luminance angular prediction modes only once and uses the result for the corresponding prediction modes. There are 33792, 8448, 2112 and 528 prediction equations in 32x32, 16x16, 8x8 and 4x4 luminance angular prediction modes, respectively. As shown in Table 2.1, using data reuse technique, the numbers of prediction equations that should be calculated for 32x32, 16x16, 8x8 and 4x4 luminance angular prediction modes are reduced to 3735, 1507, 593 and 201, respectively.

Table 2.1 Prediction Equation Reductions by Data Reuse

	4x4 PU	**8x8 PU**	**16x16 PU**	**32x32 PU**	**32x32 CU**
# of P. Equations	528	2112	8448	33792	135168
# of P. Equations with Data Reuse	201	593	1507	3735	14848
Reduction (%)	61.93	71.92	82.16	88.94	89.02

A 32x32 CU includes one 32x32 PU, four 16x16 PUs, sixteen 8x8 PUs and sixty four 4x4 PUs. As shown in Figure 2.2, an 8x8 PU and some of the 4x4 PUs have common neighboring pixels. They also have common prediction equations. 4x4, 8x8, 16x16 and 32x32 PUs also have common neighboring pixels and common prediction equations. Therefore, data reuse technique is used for calculating predicted pixels of a 32x32 PU and predicted pixels of the corresponding four 16x16 PUs, sixteen 8x8 PUs and sixty four 4x4 PUs. In this way, the number of prediction equations that should be calculated for a 32x32 CU is reduced from 135168 to 14848.

In this thesis, a novel technique is proposed for reducing amount of computations performed by HEVC intra prediction algorithm. The proposed technique reorganizes the HEVC intra prediction equations by utilizing the fact that the sum of the coefficients used in each HEVC angular intra prediction equation is 32. This reduces the amount of computations performed by 4x4, 8x8, 16x16 and 32x32 luminance angular prediction modes. It does not affect the PSNR and bit rate.

The original version of each intra prediction equation requires two multiplications with constants. Both constants are between 1 and 31. The sum of both constants is 32. Reorganized version of each intra prediction equation requires two multiplications with constants. One constant is always 32. The other constant is between 1 and 16. Multiplications with constants are implemented using addition and shift operations. The reorganized intra prediction equations require less number of addition and shift operations than the original ones.

An HEVC intra prediction equation and its reorganized version are shown in equations (2.4a) and (2.5a), respectively. As shown in equation (2.4b), original intra prediction equation requires six addition and five shift operations. As shown in equations (2.5b) and (2.5c), its reorganized version requires two addition, two subtraction and three shift operations. Another HEVC intra prediction equation and its reorganized version are shown in equations (2.6a) and (2.7a), respectively. As shown in equation (2.6b), original intra prediction equation requires six addition and five shift operations. As shown in equations (2.7b) and (2.7c), its reorganized version requires one addition, two subtraction and two shift operations.

$$(9 * A + 23 * B + 16) \gg 5 \tag{2.4a}$$

$$(A + (A \ll 3) + B + (B \ll 1) + (B \ll 2) + (B \ll 4) + 16) \gg 5 \tag{2.4b}$$

$$(32 * B - 9 * (B - A) + 16) \gg 5 \tag{2.5a}$$

$$temp = B - A \tag{2.5b}$$

$$((B \ll 5) - (temp + (temp \ll 3)) + 16) \gg 5 \tag{2.5c}$$

$$(A + 31 * B + 16) \gg 5 \tag{2.6a}$$

$$(A + B + (B \ll 1) + (B \ll 2) + (B \ll 3) + (B \ll 4) + 16) \gg 5 \tag{2.6b}$$

$$(32 * B - (B - A) + 16) \gg 5 \tag{2.7a}$$

$$temp = B - A \tag{2.7b}$$

$$((B \ll 5) - (temp) + 16) \gg 5 \tag{2.7c}$$

Numbers of addition and shift operations required for original HEVC intra prediction algorithm and HEVC intra prediction algorithm with reorganized equations for all the PUs in a 32x32 CU after using data reuse technique are shown in Table 2.2. The

total numbers of addition and shift operations are calculated by adding the numbers of addition and shift operations required for each intra angular prediction equation for all the PUs in a 32x32 CU. Subtraction operations are counted as addition operations. The proposed technique reduces numbers of addition and shift operations by 40.3% and 49.8%, respectively.

Table 2.2 Addition and Shift Reductions by the Proposed Technique

	Original	Reorganized	Reduction (%)
# of Addition	75348	45024	40.3
# of Shift	84932	42652	49.8

2.2.2 Proposed HEVC Intra Prediction Hardware

The proposed HEVC intra prediction hardware implementing angular prediction modes for all PU sizes (4x4, 8x8, 16x16 and 32x32) including data reuse and the proposed technique is shown in Figure 2.3. There are ten pipelined datapaths. Each datapath calculates the result of one intra prediction equation in each clock cycle. Therefore, ten parallel datapaths calculate the results of ten intra prediction equations in each clock cycle.

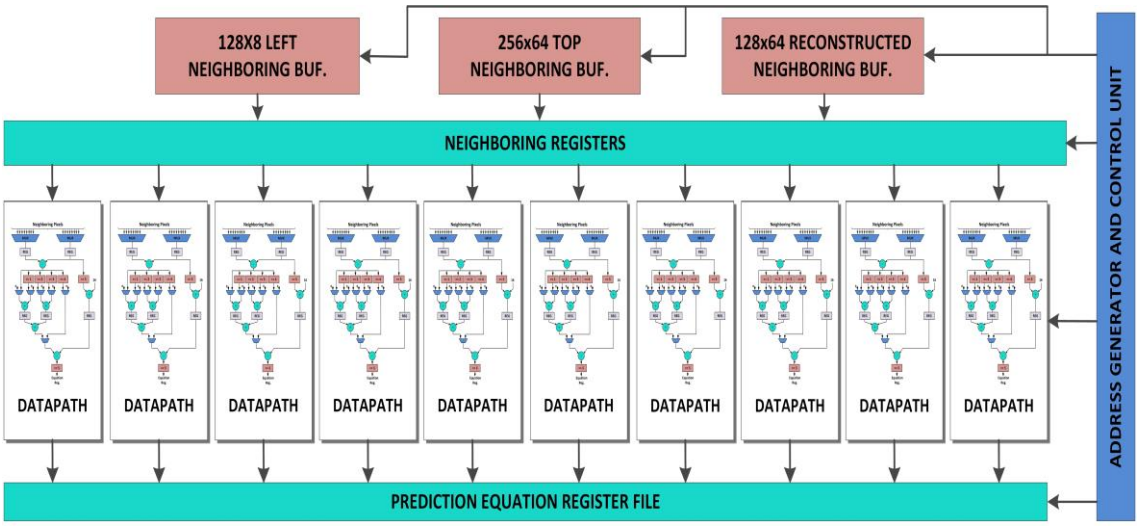

Figure 2.3 Proposed HEVC Intra Prediction Hardware

Three local neighboring buffers are used to store neighboring pixels in the previously coded and reconstructed neighboring PUs. After a PU in the current CU is coded and reconstructed, the neighboring pixels in this PU are stored in the corresponding

buffers. These on chip neighboring buffers reduce the required off-chip memory bandwidth. The predicted pixels are stored in the prediction equation register file.

A 32x32 CU, which includes one 32x32 PU, four 16x16 PUs, sixteen 8x8 PUs and sixty four 4x4 PUs, has 528 neighboring pixels. Storing all 528 neighboring pixels in 528 registers would increase the hardware area. In order to reduce the hardware area, 32x32 CU is split into 8x8 blocks and prediction equations, regardless of their PU sizes, are divided into groups based on the pixels they use. The prediction equations using pixels from the same 8x8 block are grouped together. In this way, only neighboring pixels of current 8x8 block and corresponding four 4x4 blocks are stored in 42 registers. After these neighboring pixel registers are loaded in 16 clock cycles, ten parallel datapaths are used to calculate the prediction equations for current 8x8 block and corresponding four 4x4 blocks.

The proposed datapath for calculating reorganized versions of HEVC intra prediction equations is shown in Figure 2.4. This datapath requires adder and shifter hardware for two multiplications with constants. One constant is always 32. The other constant is between 1 and 16. The datapath necessary for calculating original versions of HEVC intra prediction equations is shown in Figure 2.5. This datapath requires adder and shifter hardware for two multiplications with constants. Both constants are between 1 and 31. Therefore, the proposed datapath requires less hardware area and consumes less power.

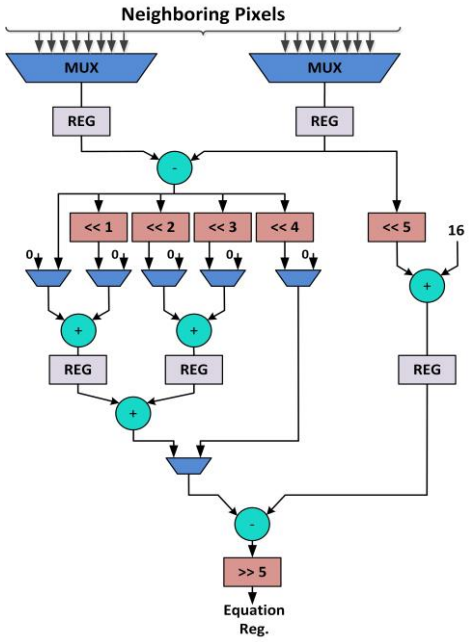

Figure 2.4 Proposed HEVC Intra Prediction Datapath

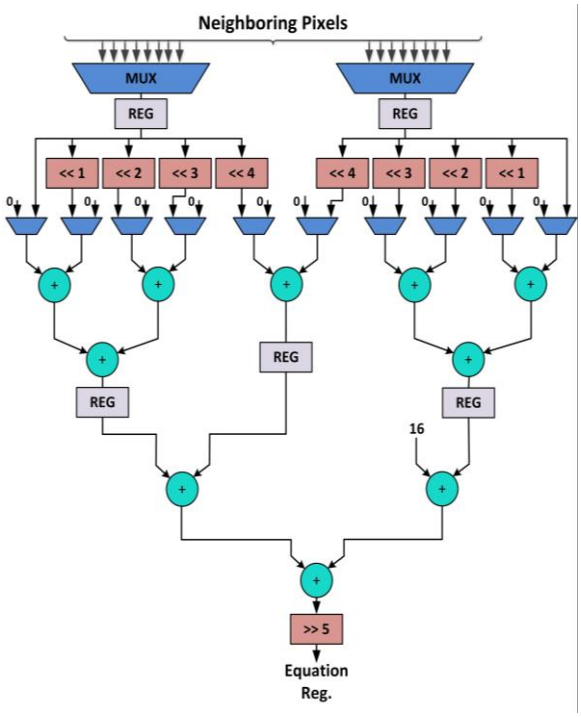

Figure 2.5 Original HEVC Intra Prediction Datapath

The proposed hardware is implemented using Verilog HDL. The Verilog RTL code is verified with RTL simulations. The RTL simulation results matched the results of HEVC intra prediction implementation in HEVC HM software encoder [34]. The Verilog RTL code is synthesized and mapped to an FPGA implemented in 40nm CMOS technology. The FPGA implementation is verified with post place and route simulations. Post place and route simulation results matched the results of HEVC intra prediction implementation in HEVC HM software encoder [34].

As shown in Figure 2.6, the FPGA implementation is also verified to work correctly on an FPGA board which includes an FPGA implemented in 40 nm CMOS technology, 512 MB external memory and interfaces such as UART and DVI. In the FPGA, processor local bus (PLB) is used for the communication between the proposed HEVC intra prediction hardware and microprocessor. The proposed FPGA implementation uses 6013 LUTs, 2006 DFFs and 4 BRAMs. It can work at 166 MHz, and it can process 40 full HD (1920x1080) video frames per second.

Verilog RTL code of the proposed HEVC intra prediction hardware is also synthesized to a 90 nm standard cell library and the resulting netlist is placed and routed. The resulting ASIC implementation can work at 250 MHz, and it can process 60 full HD

(1920x1080) video frames per second. Its gate count is 16.1K, according to NAND (2x1) gate area excluding on-chip memory.

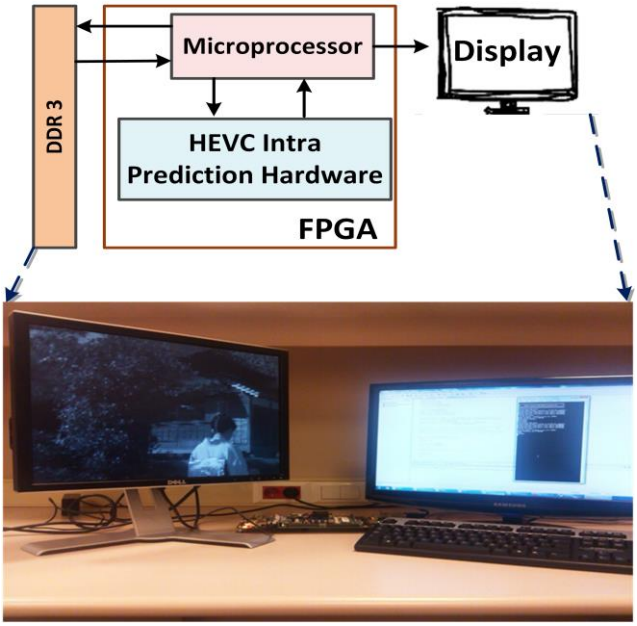

Figure 2.6 FPGA Implementation of HEVC Intra Prediction Hardware

Power consumption of the proposed FPGA implementation is estimated using a gate level power estimation tool. Post place and route timing simulations are performed for Tennis and Kimono videos at 100 MHz [35], and signal activities are stored in VCD files. These VCD files are used for estimating power consumption of the FPGA implementation. The power and energy consumption results of the FPGA implementation for one frame of each video quantized with three different quantization parameters (QP) are shown in Table 2.3 and Table 2.4.

Table 2.3 Energy Consumption Reductions for Kimono (1920x1080)

	Original HEVC Intra Prediction Hardware			**Proposed HEVC Intra Prediction Hardware**		
QP	**28**	**35**	**42**	**28**	**35**	**42**
Time (ms)	40.78	40.78	40.78	40.78	40.78	40.78
Clock (mW)	27.91	27.91	27.91	23.02	23.02	23.02
Signal (mW)	21.74	21.61	21.57	17.94	17.87	17.42
Logic (mW)	18.53	18.36	18.31	12.52	12.44	11.70
BRAM (mW)	2.54	2.54	2.54	2.54	2.54	2.54
Power (mW)	70.72	70.72	70.33	56.02	55.87	54.68
Energy (uJ)	2884.5	2884.5	2868.6	2284.9	2278.8	2230.3
Energy Reduction				20.79 %	21.00 %	22.25 %

Table 2.4 Energy Consumption Reductions for Tennis (1920x1080)

	Original HEVC Intra Prediction Hardware			Proposed HEVC Intra Prediction Hardware		
QP	**28**	**35**	**42**	**28**	**35**	**42**
Time (ms)	40.78	40.78	40.78	40.78	40.78	40.78
Clock (mW)	27.91	27.91	27.91	23.02	23.02	23.02
Signal (mW)	22.03	21.93	22.20	17.49	17.13	17.61
Logic (mW)	19.27	19.15	19.53	11.76	11.22	11.99
BRAM (mW)	2.54	2.54	2.54	2.54	2.54	2.54
Power (mW)	71.75	71.53	72.18	54.81	53.91	55.16
Energy (uJ)	2926.5	2917.6	2944.1	2235.6	2198.9	2249.9
Energy Reduction				23.61 %	24.63 %	23.58 %

The time it takes for the FPGA implementation to process one frame is shown in the tables. Original HEVC intra prediction hardware does not use the proposed computation and energy reduction technique. Therefore, it uses the original HEVC intra prediction datapath shown in Figure 2.5. Both original and proposed HEVC intra prediction hardware calculate the result of one intra prediction equation in each clock cycle. The proposed technique did not affect the critical path of the HEVC intra prediction hardware. Therefore, the time it takes to process one frame is the same for both original and proposed HEVC intra prediction hardware.

However, as it can be seen from Figure 2.4 and Figure 2.5, since the proposed HEVC intra prediction hardware performs less addition and shift operations in one clock cycle than original HEVC intra prediction hardware, it has smaller hardware area. Therefore, it consumes up to 24.63% less energy than original HEVC intra prediction hardware. Since HEVC intra prediction hardware is used as part of an HEVC video encoder, only internal power consumption is considered, input and output power consumptions are ignored. Therefore, power consumption of the FPGA implementation can be divided into four main categories; clock power, logic power, signal power and BRAM power.

Comparisons of the FPGA and ASIC implementations of proposed HEVC intra prediction hardware with the FPGA and ASIC implementations of HEVC intra prediction hardware proposed in the literature are shown in Table 2.5 and Table 2.6, respectively. The area of the proposed hardware is much smaller than the ones proposed in [24]-[32]. Power consumptions of the hardware implementations proposed in [24]-[31] are not reported. The proposed hardware consumes less power than the one proposed in [32].

Table 2.5 Comparison of FPGA Implementations

	[24]	[25]	[26]	[27]	[33]	Proposed
Technology	65 nm	28 nm	40 nm	40 nm	40 nm	40 nm
DFF	5.5 K	22 K	110 K	6934	849	2006
LUT	14 K	43 K	170 K	13409	2381	6013
BRAM	---	94	---	---	4	4
Max Freq. (MHz)	110	150	219	162	150	166
Frames per Sec.	30 3840x2160	---	24 3840x2160	---	30 1920x1080	40 1920x1080
PU Size	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8	4, 8, 16, 32

Table 2.6 Comparison of ASIC Implementations

	[28]	[29]	[30]	[31]	[32]	[33]	Proposed
Technology	90 nm	40 nm	90 nm	130 nm	90 nm	90 nm	90 nm
Gate Count	127.3 K	27 K	76.8 K	324 K	712.2 K	5.4 K	16.1 K
Max Freq. (MHz)	200	200	270	400	357	150	250
Frames per Sec.	30 3840x2160	---	---	60 1920x1080	46 2560x1600	30 1920x1080	60 1920x1080
Memory	6 KB	4.9 KB	5.6 KB	---	---	---	3 KB
Power Dissipation	---	---	---	---	92.1 mW	23.2 mW	28.5 mW
PU Size	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8	4, 8, 16, 32

The proposed HEVC intra prediction hardware implementation performs intra prediction for all PU sizes. Since the HEVC intra prediction hardware implementation proposed in [33] performs intra prediction only for 4x4 and 8x8 PU sizes, it has smaller area and consumes less power than the proposed HEVC intra prediction hardware.

Some of the HEVC intra prediction hardware implementations have higher performance than the proposed HEVC intra prediction hardware implementation at the expense of much larger hardware area. The frames per second performance of the HEVC intra prediction hardware implementation proposed in [27] is not reported. Since the HEVC intra prediction hardware implementations in [25, 29, 30] are proposed for an HEVC decoder, their frames per second performances for an HEVC encoder are not reported.

2.3 DSP Block Based FPGA Implementation of HEVC Intra Prediction

A computation and energy reduction technique for HEVC intra prediction is proposed in [18]. This technique reorganizes the HEVC intra prediction equations by utilizing the fact that the sum of the coefficients used in each HEVC angular intra prediction equation is 32. This reduces the amount of computations performed by 4x4,

8x8, 16x16 and 32x32 luminance angular prediction modes. It does not affect the PSNR and bit rate.

Xilinx FPGAs have built-in full-custom DSP blocks which can perform constant multiplications faster and with less energy than adders and shifters. A DSP block can be used to perform different constant multiplications by providing proper constant value to its input. Therefore, it is more efficient to implement constant multiplications using DSP blocks instead of using adders and shifters in an FPGA implementation.

In this thesis, an efficient FPGA implementation of HEVC intra prediction for angular prediction modes of all PU sizes (4x4, 8x8, 16x16 and 32x32) is proposed. The proposed FPGA implementation uses the computation and energy reduction technique for HEVC intra prediction proposed in [18]. However, it implements intra angular prediction equations using DSP blocks in FPGA instead of using adders and shifters. In this way, one HEVC intra angular prediction equation is implemented using only one DSP block instead of using two DSP blocks and two adders.

The proposed FPGA implementation can work at 227 MHz in a Xilinx Virtex 6 FPGA. It, in the worst case, can process 55 Full HD (1920x1080) video frames per second. The proposed FPGA implementation has up to 15.97% less energy consumption than the FPGA implementation of HEVC intra prediction using the computation and energy reduction technique proposed in [18] and adders and shifters. The proposed FPGA implementation has up to 34.66% less energy consumption than the FPGA implementation of HEVC intra prediction using original prediction equations and DSP blocks.

Several HEVC intra prediction hardware are proposed in the literature [18], [24]-[27], [33]. They are compared with the proposed HEVC intra prediction hardware.

The proposed HEVC intra prediction hardware implementing angular prediction modes for all PU sizes (4x4, 8x8, 16x16 and 32x32) using the computation and energy reduction technique proposed in [18] and DSP blocks is shown in Figure 2.7. There are ten pipelined datapaths. Each datapath calculates the result of one intra prediction equation in each clock cycle. Therefore, ten parallel datapaths calculate the results of ten intra prediction equations in each clock cycle.

Three local neighboring buffers are used to store neighboring pixels in the previously coded and reconstructed neighboring PUs. After a PU in the current CU is coded and reconstructed, the neighboring pixels in this PU are stored in the corresponding

buffers. These on chip neighboring buffers reduce required off-chip memory bandwidth. The predicted pixels are stored in the prediction equation register file.

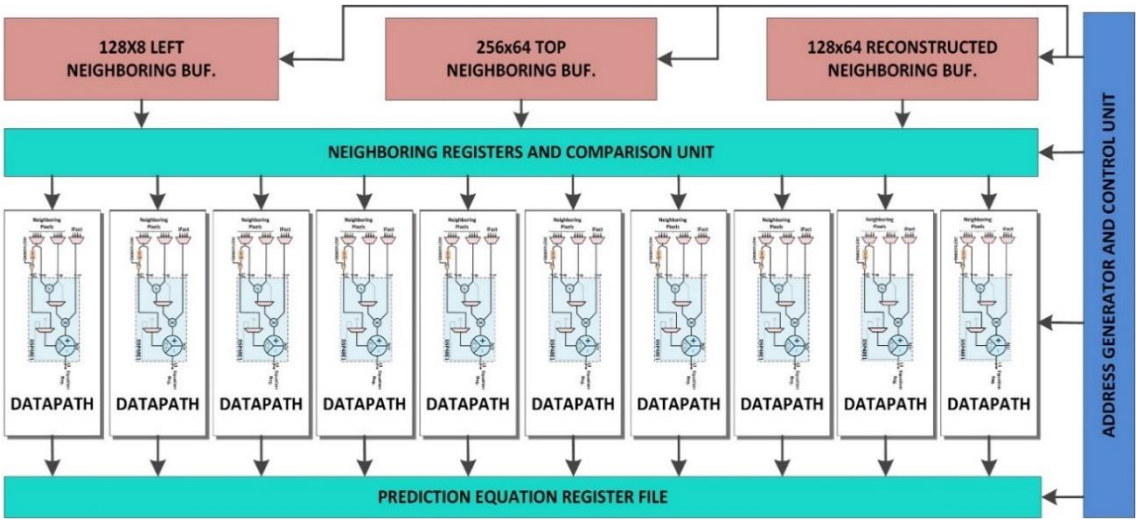

Figure 2.7 Proposed HEVC Intra Prediction Hardware

A 32x32 CU has 528 neighboring pixels. Storing all 528 neighboring pixels in 528 registers would increase the hardware area. In order to reduce the hardware area, 32x32 CU is split into 8x8 blocks and prediction equations, regardless of their PU sizes, are divided into groups based on the pixels they use. The prediction equations using pixels from the same 8x8 block are grouped together. In this way, only neighboring pixels of current 8x8 block and corresponding four 4x4 blocks are stored in 42 registers. After these neighboring pixel registers are loaded in 16 clock cycles, ten parallel datapaths are used to calculate the prediction equations for current 8x8 block and corresponding four 4x4 blocks.

In an FPGA implementation, multiplication operations in the intra prediction equations can be implemented more efficiently using DSP blocks instead of using adders and shifters. Structure of a DSP48E1 block is shown in Figure 2.8. If constant multiplications are implemented using adders and shifters, 10 adders and 10 multiplexers are necessary to implement one original intra prediction equation [18]. If constant multiplications are implemented using DSP blocks, as shown in Figure 2.9, two DSP blocks and two adders are necessary to implement one original intra prediction equation.

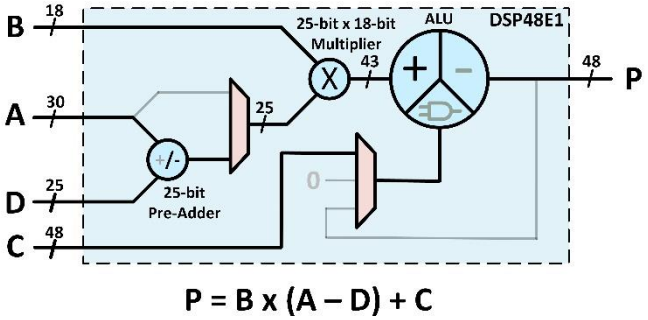

Figure 2.8 Structure of a DSP48E1 Block

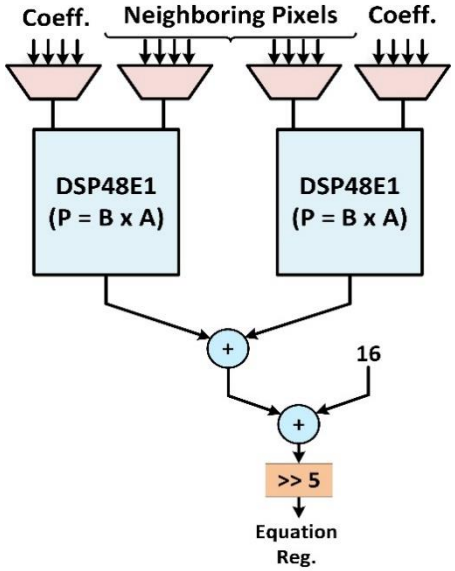

Figure 2.9 Original HEVC Intra Prediction Datapath

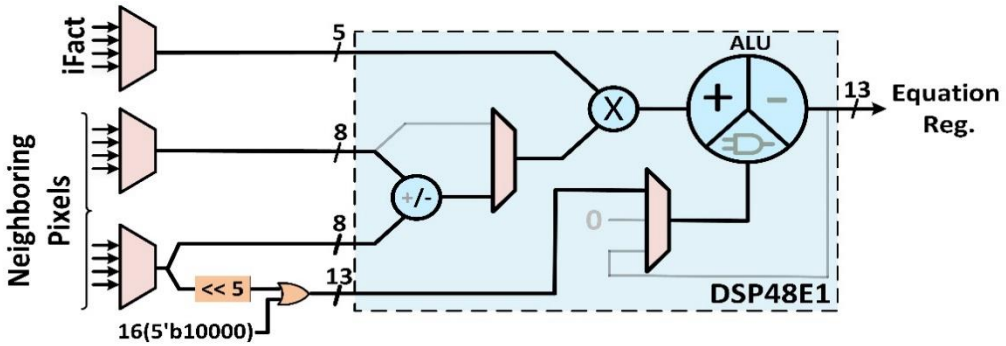

Figure 2.10 Proposed HEVC Intra Prediction Datapath

However, as shown in Figure 2.10, one reorganized intra prediction equation can be implemented using only one DSP block. The DSP block is configured to perform multiplication and addition operations. For example, reorganized intra prediction

equation shown in (2.5a) is implemented using a DSP block as follows. $(9 * (A - B))$ is implemented using part of the DSP block implementing $B * (A \pm D)$. One neighboring pixel is shifted left by 5 and ORed with 16 to implement $(32 * B + 16)$ and the result is given to C input of DSP block. Since the last 5 bits of $32 * B$ is zero, $(32 * B + 16)$ can be implemented by changing 5th bit of $32 * B$ from zero to one.

In this thesis, an HEVC intra prediction hardware implementing angular prediction modes for all PU sizes (4x4, 8x8, 16x16 and 32x32) using the original intra prediction equations and DSP blocks is also designed for comparison. Both HEVC intra prediction hardware designs are implemented using Verilog HDL. The Verilog RTL codes are verified with RTL simulations. RTL simulation results matched the results of HEVC intra prediction implementation in HEVC HM software encoder [34].

The Verilog RTL codes are synthesized and mapped to a Xilinx XC6VLX75T FF1759 FPGA with speed grade 3 using Xilinx ISE 14.7. FPGA implementations are verified with post place and route simulations. Post place and route simulation results matched the results of HEVC intra prediction implementation in HEVC HM software encoder [34].

FPGA implementation results of HEVC intra prediction hardware using original intra prediction equations and adders and shifters (ORG_AS) [18], reorganized intra prediction equations and adders and shifters (REORG_AS) [18], original intra prediction equations and DSP blocks (ORG_DSP), reorganized intra prediction equations and DSP blocks (REORG_DSP) are shown in Table 2.7.

Table 2.7 Implementation Results

	ORG_AS [18]	**REORG_AS [18]**	**ORG_DSP**	**REORG_DSP**
FPGA	Xilinx Virtex 6	Xilinx Virtex 6	Xilinx Virtex 6	Xilinx Virtex 6
DFF	2567	2006	1167	1168
LUT	5521	6013	4510	4425
BRAM	4	4	4	4
DSP48E1	---	---	20	10
Max. Freq. (MHz)	166	166	212	227
Frames per Second	40 1920x1080	40 1920x1080	52 1920x1080	55 1920x1080
PU Size	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32

Power consumptions of all FPGA implementations are estimated using Xilinx XPower Analyzer tool. Post place and route timing simulations are performed for Tennis

and Kimono videos at 100 MHz [35], and signal activities are stored in VCD files. These VCD files are used for estimating power consumptions of FPGA implementations.

Energy consumption results of all FPGA implementations for one frame of each video quantized with three different quantization parameters (QP) are shown in Figure 2.11. The proposed FPGA implementation of HEVC intra prediction using the computation and energy reduction technique proposed in [18] and DSP blocks has up to 15.97% less energy consumption than the FPGA implementation of HEVC intra prediction using the computation and energy reduction technique proposed in [18] and adders and shifters. The proposed FPGA implementation of HEVC intra prediction using the computation and energy reduction technique proposed in [18] and DSP blocks has up to 34.66% less energy consumption than the FPGA implementation of HEVC intra prediction using original prediction equations and DSP blocks.

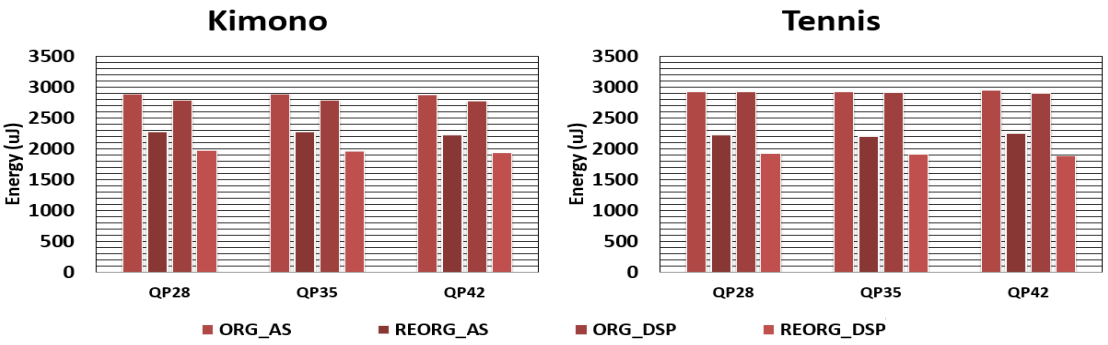

Figure 2.11 Energy Consumption Results

Comparison of the proposed FPGA implementation of HEVC intra prediction using the computation and energy reduction technique proposed in [18] and DSP blocks with the FPGA implementations of HEVC intra prediction hardware proposed in the literature is shown in Table 2.8. Area of the proposed FPGA implementation is smaller than the ones proposed in [18], [24]-[27]. Power consumptions of the HEVC intra prediction hardware proposed in [24]-[27] are not reported. The proposed FPGA implementation consumes less power than the one proposed in [18]. Since the HEVC intra prediction hardware proposed in [33] performs intra prediction only for 4x4 and 8x8 PU sizes, it has smaller area and consumes less power than the proposed hardware.

Some of the HEVC intra prediction hardware have higher performance than the proposed HEVC intra prediction hardware at the expense of much larger hardware area. Frames per second performance of the HEVC intra prediction hardware proposed in [27]

is not reported. Since the HEVC intra prediction hardware in [25] is proposed for an HEVC decoder, its frames per second performance for an HEVC encoder is not reported.

Table 2.8 Comparison of FPGA Implementations

	[18]	**[24]**	**[25]**	**[26]**	**[27]**	**[33]**	**Proposed**
FPGA	Xilinx Virtex 6	65 nm FPGA	Xilinx Zynq 7045	Xilinx Virtex 6	Altera Arria II GX	Xilinx Virtex 6	Xilinx Virtex 6
DFF	2006	5.5 K	22 K	110 K	6934	849	1168
LUT	6013	14 K	43 K	170 K	13409	2381	4425
BRAM	4	---	94	---	---	4	4
DSP48E1	---	---	---	---	8	---	10
Max. Freq. (MHz)	166	110	150	219	162	150	227
Frames per Second	40 1920x1080	30 3840x2160	---	24 3840x2160	---	30 1920x1080	55 1920x1080
PU Size	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8	4, 8, 16, 32

CHAPTER III

VVC INTRA PREDICTION HARDWARE

3.1 VVC Intra Prediction Algorithm

VVC intra prediction algorithm predicts pixels of a PU using neighboring pixels in neighboring PUs. 4x4, 8x8, 16x16, 32x32, 64x64 PU sizes are used for luminance components of frames. VVC has 65 intra angular prediction modes (mode) for each PU size. Prediction angles (angle) corresponding to each prediction mode are shown in Figure 3.1. VVC also has DC and planar prediction modes for each PU size. Neighboring pixels of an 8x8 PU and four 4x4 PUs are shown in Figure 3.2.

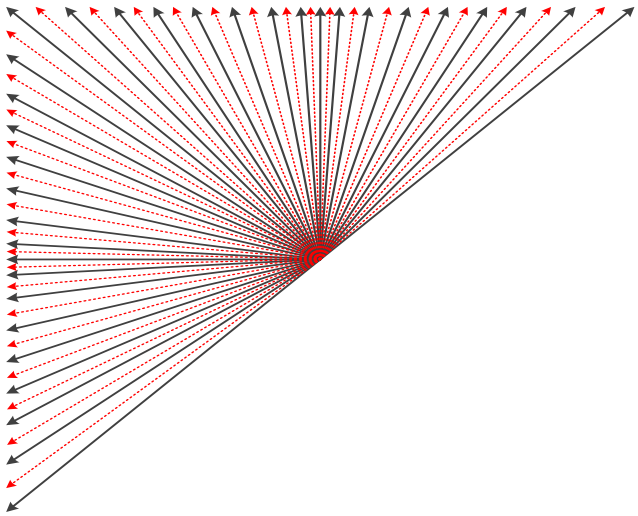

Figure 3.1 VVC Intra Prediction Angles

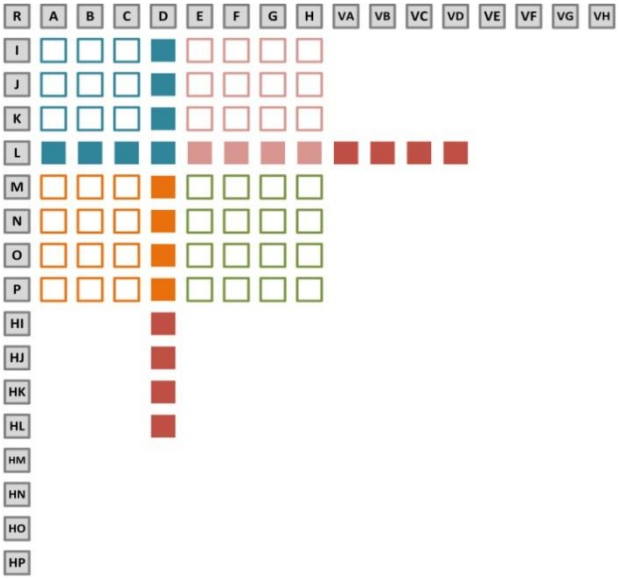

Figure 3.2 Neighboring Pixels

Table 3.1 Cubic and Gaussian Filter Coefficients

	Filter	Coefficients			
Cubic Filters	1	0	256	0	0
	2	-3	252	8	-1
	3	-5	247	17	-3
	4	-7	242	25	-4
	5	-9	236	34	-5
	6	-10	230	43	-7
	7	-12	224	52	-8
	8	-13	217	61	-9
	9	-14	210	70	-10
	10	-15	203	79	-11
	11	-16	195	89	-12
	12	-16	187	98	-13
	13	-16	179	107	-14
	14	-16	170	116	-14
	15	-17	162	126	-15
	16	-16	153	135	-16
	17	-16	144	144	-16
Gaussian Filters	18	47	161	47	1
	19	43	161	51	1
	20	40	160	54	2
	21	37	159	58	2
	22	34	158	62	2
	23	31	156	67	2
	24	28	154	71	3
	25	26	151	76	3
	26	23	149	80	4
	27	21	146	85	4
	28	19	142	90	5
	29	17	139	94	6
	30	16	135	99	6
	31	14	131	104	7
	32	13	127	108	8
	33	11	123	113	9
	34	10	118	118	10

17 different 4-tap cubic filters and 17 different 4-tap gaussian filters are used as intra prediction equations. Coefficients of these 4-tap filters are shown in Table 3.1. Cubic filters are used for 4x4 and 8x8 prediction units. Gaussian filters are used for 16x16, 32x32 and 64x64 prediction units.

VVC intra prediction algorithm determines reference pixel array (rparray) which consists of pixels that will be used in intra prediction equations of the corresponding prediction mode and PU size. Reference pixel array is filled with above neighboring pixels if prediction mode is more than or equal to 34. However, if prediction angle is less than zero, its first four pixels are filled with left neighboring pixels. Reference pixel array is filled with left neighboring pixels if prediction mode is less than 34. However, if prediction angle is less than zero, its first four pixels are filled with above neighboring pixels.

VVC intra prediction algorithm calculates deltaint as shown in equation (3.1a). It calculates deltafract as shown in equation (3.1b). deltaint is used for determining positions of pixels in reference pixel array that will be used in intra prediction equations. Four pixels used in intra prediction equations are adjacent pixels in reference pixel array, but they may not be adjacent in video frame. These four pixels are selected as shown in equations (3.2a)-(3.2e), where rp[0], rp[1], rp[2] and rp[3] are the selected pixels from reference pixel array. If rp[1] is the left-most pixel in reference pixel array, rp[0] is equal to rp[1]. If rp[2] is the right-most pixel in the reference pixel array, rp[3] is equal to rp[2]. PU size is used for determining whether cubic or gaussian filters will be used. deltafract is used for determining which 4-tap filter among 17 4-tap filters will be used.

$$deltaint = \big((y+1) * angle\big) \gg 5 \tag{3.1a}$$

$$deltafract = \big((y+1) * angle\big) \,\&\, 31 \tag{3.1b}$$

$$rparrayindex = x + deltaint + 1 \tag{3.2a}$$

$$\mathrm{rp}[1] = rparray[rparrayindex] \tag{3.2b}$$

$$rp[2] = rparray[rparrayindex + 1] \tag{3.2c}$$

$$rp[0] = (x == 0)?\, rp[1] \,:\, rparray[rparrayindex - 1] \tag{3.2d}$$

$$rp[3] = (x == (width - 1))\,?\, rp[2] : rparray[rparrayindex + 2] \tag{3.2e}$$

$$x = 0\ to\ (PU_{size} - 1), y = 0\ to\ (PU_{size} - 1)$$

Reference pixel array and prediction equations for 8x8 intra angular prediction mode 9 with prediction angle -13 are shown in equations (3.3a) and (3.3b), respectively.

$$rparray = [0, 0 ,0, 0, 0, O, M, J, R, A, B, C, D, E, F, G, H, 0, 0, 0, 0, 0, 0, 0, 0] \quad (3.3a)$$

$$\begin{gathered} pp[0,0] = pp[1,0] = (-17C + 162B + 126A - 15A) \gg 8 \\ pp[2,0] = pp[3,0] = (-17B + 162A + 126R – 15R) \gg 8 \\ pp[4,0] = pp[5,0] = pp[6,0] = (-17A + 162R + 126J – 15J) \gg 8 \\ pp[7,0] = (-17R + 162J + 126M – 15M) \gg 8 \end{gathered} \quad (3.3b)$$

$$\begin{gathered} pp[0,1] = pp[1,1] = (-10A + 230B + 43C - 7D) \gg 8 \\ pp[2,1] = pp[3,1] = (-10R + 230A + 43B - 7C) \gg 8 \\ pp[4,1] = pp[5,1] = pp[6,1] = (-10J + 230R + 43A - 7B) \gg 8 \\ pp[7,1] = (-10M + 230J + 43R - 7A) \gg 8 \end{gathered}$$

$$\begin{gathered} pp[0,2] = pp[1,2] = (-14E + 210D + 70C – 10B) \gg 8 \\ pp[2,2] = pp[3,2] = (-14D + 210C + 70B – 10A) \gg 8 \\ pp[4,2] = pp[5,2] = pp[6,2] = (-14C + 210B + 70A – 10R) \gg 8 \\ pp[7,2] = (-14B + 210A + 70R – 10J) \gg 8 \end{gathered}$$

$$\begin{gathered} pp[0,3] = pp[1,3] = (-16C + 187D + 98E – 13F) \gg 8 \\ pp[2,3] = pp[3,3] = (-16B + 187C + 98D – 13E) \gg 8 \\ pp[4,3] = pp[5,3] = pp[6,3] = (-16A + 187B + 98C – 13D) \gg 8 \\ pp[7,3] = (-16R + 187A + 98B – 13C) \gg 8 \end{gathered}$$

$$\begin{gathered} pp[0,4] = pp[1,4] = (-5G + 247F + 17E – 3D) \gg 8 \\ pp[2,4] = pp[3,4] = (-5F + 247E + 17D – 3C) \gg 8 \\ pp[4,4] = pp[5,4] = pp[6,4] = (-5E + 247D + 17C – 3B) \gg 8 \\ pp[7,4] = (-5D + 247C + 17B – 3A) \gg 8 \end{gathered}$$

$$\begin{gathered} pp[0,5] = pp[1,5] = (-16H + 153G + 135F – 16E) \gg 8 \\ pp[2,5] = pp[3,5] = (-16G + 153F + 135E – 16D) \gg 8 \\ pp[4,5] = pp[5,5] = pp[6,5] = (-16F + 153E + 135D – 16C) \gg 8 \\ pp[7,5] = (-16E + 153D + 135C – 16B) \gg 8 \end{gathered}$$

$$\begin{gathered} pp[0,6] = pp[1,6] = (-9F + 236G + 34H) \gg 8 \\ pp[2,6] = pp[3,6] = (-9E + 236F + 34G – 5H) \gg 8 \\ pp[4,6] = pp[5,6] = pp[6,6] = (-9D + 236E + 34F – 5G) \gg 8 \\ pp[7,6] = (-9C + 236D + 34E – 5F) \gg 8 \end{gathered}$$

$$\begin{gathered} pp[0,7] = pp[1,7] = (79H – 11G) \gg 8 \\ pp[2,7] = pp[3,7] = (-15H + 203H + 79G – 11F) \gg 8 \\ pp[4,7] = pp[5,7] = pp[6,7] = (-15G + 203G + 79F – 11E) \gg 8 \\ pp[7,7] = (-15F + 203F + 79E – 11D) \gg 8 \end{gathered}$$

3.2 Reconfigurable Intra Angular Prediction Hardware for VVC

Two VVC reconfigurable intra prediction hardware are proposed. They implement 65 VVC intra angular prediction modes for 4x4, 8x8, 16x16, 32x32 prediction units. The first reconfigurable hardware (RECON_AS) implements multiplications with constants using adders and shifters instead of using multipliers. Therefore, it can be used in ASIC

implementations of VVC encoders. It uses thirty reconfigurable datapaths. Each RECON_AS datapath can calculate any 4-tap gaussian and cubic filter used in VVC intra angular prediction. It is configured by a filter selection signal in each clock cycle.

FPGAs have built-in full-custom DSP blocks which can perform constant multiplications faster and with less energy than adders and shifters. A DSP block can be used to perform different constant multiplications by providing proper constant value to its input. Therefore, it is more efficient to implement constant multiplications using DSP blocks instead of using adders and shifters in an FPGA implementation.

The second reconfigurable hardware (RECON_DSP) implements multiplications with constants using DSP blocks in FPGA instead of using adders and shifters. Therefore, it can be used in FPGA implementations of VVC encoders. It uses thirty reconfigurable datapaths. Each RECON_DSP datapath uses four DSP blocks. It can calculate any 4-tap gaussian and cubic filter used in VVC intra angular prediction. It is configured by changing DSP inputs in each clock cycle.

RECON_AS and RECON_DSP VVC intra prediction hardware are implemented with Verilog HDL. The Verilog codes are mapped to a 28 nm FPGA and a 90 nm standard cell library. RECON_AS and RECON_DSP FPGA implementations work at 108 and 105 MHz, respectively. They process 30 full HD (1920x1080) video frames per second. RECON_AS and RECON_DSP ASIC implementations work at 218 and 208 MHz, and they process 62 full HD and 59 full HD video frames per second, respectively.

RECON_AS ASIC implementation has up to 12.8% less energy consumption than RECON_DSP ASIC implementation. Therefore, RECON_AS can be used in ASIC implementations of VVC encoders. RECON_DSP FPGA implementation has up to 30.2% less energy consumption than RECON_AS FPGA implementation. Therefore, RECON_DSP can be used in FPGA implementations of VVC encoders.

In the literature, there is no VVC intra prediction hardware. However, there are HEVC intra prediction hardware [18, 24, 26, 28, 36]. RECON_AS and RECON_DSP VVC intra prediction hardware are compared with them.

In VVC, intra angular prediction modes of a PU have identical prediction equations. Intra angular prediction modes of different PU sizes have identical prediction equations as well. In this thesis, data reuse technique is used to calculate identical prediction equations only once and use the results for the corresponding prediction modes. Prediction equations calculated with and without data reuse are shown in Table 3.2 and Table 3.3.

Table 3.2 Cubic Filter Prediction Equations

	4x4 Pred. Unit	**8x8 Pred. Unit**	**32x32 Coding Unit**
Prediction Equations	1040	4160	133120
Prediction Equations with Data Reuse	405	1042	29478
Reduction (%)	61.06	74.95	77.85

Table 3.3 Gaussian Filter Prediction Equations

	16x16 Pred. Unit	**32x32 Pred. Unit**	**32x32 Coding Unit**
Prediction Equations	16680	66560	133120
Prediction Equations with Data Reuse	2597	6641	11810
Reduction (%)	84.43	90.02	91.13

There are 4*4*65 = 1040 intra angular prediction equations for 4x4 PU size. There are 4160, 16680, 66560 intra angular prediction equations for 8x8, 16x16, 32x32 PU sizes, respectively. Data reuse technique reduced numbers of intra angular prediction equations for 4x4, 8x8, 16x16, 32x32 PU sizes to 405, 1042, 2597 and 6641, respectively.

There are 133120 cubic filter prediction equations for sixteen 8x8 PUs and sixty-four 4x4 PUs in a 32x32 CU. Data reuse technique reduced number of these cubic filter prediction equations to 29478. There are 133120 gaussian filter prediction equations for one 32x32 PU and four 16x16 PUs in a 32x32 CU. Data reuse technique reduced number of these gaussian filter prediction equations to 11810.

The proposed VVC reconfigurable intra prediction hardware is shown in Figure 3.3 (a). It implements 65 angular prediction modes for 4x4, 8x8, 16x16, 32x32 PU sizes. It uses data reuse technique. It has thirty pipelined reconfigurable datapaths (RDP). Each RDP calculates an intra prediction equation in a clock cycle. Thirty RDPs calculate thirty intra prediction equations in a clock cycle. Neighboring pixels in neighboring PUs are stored in left, top and reconstructed neighboring buffers. Predicted pixels are stored in prediction equation register file.

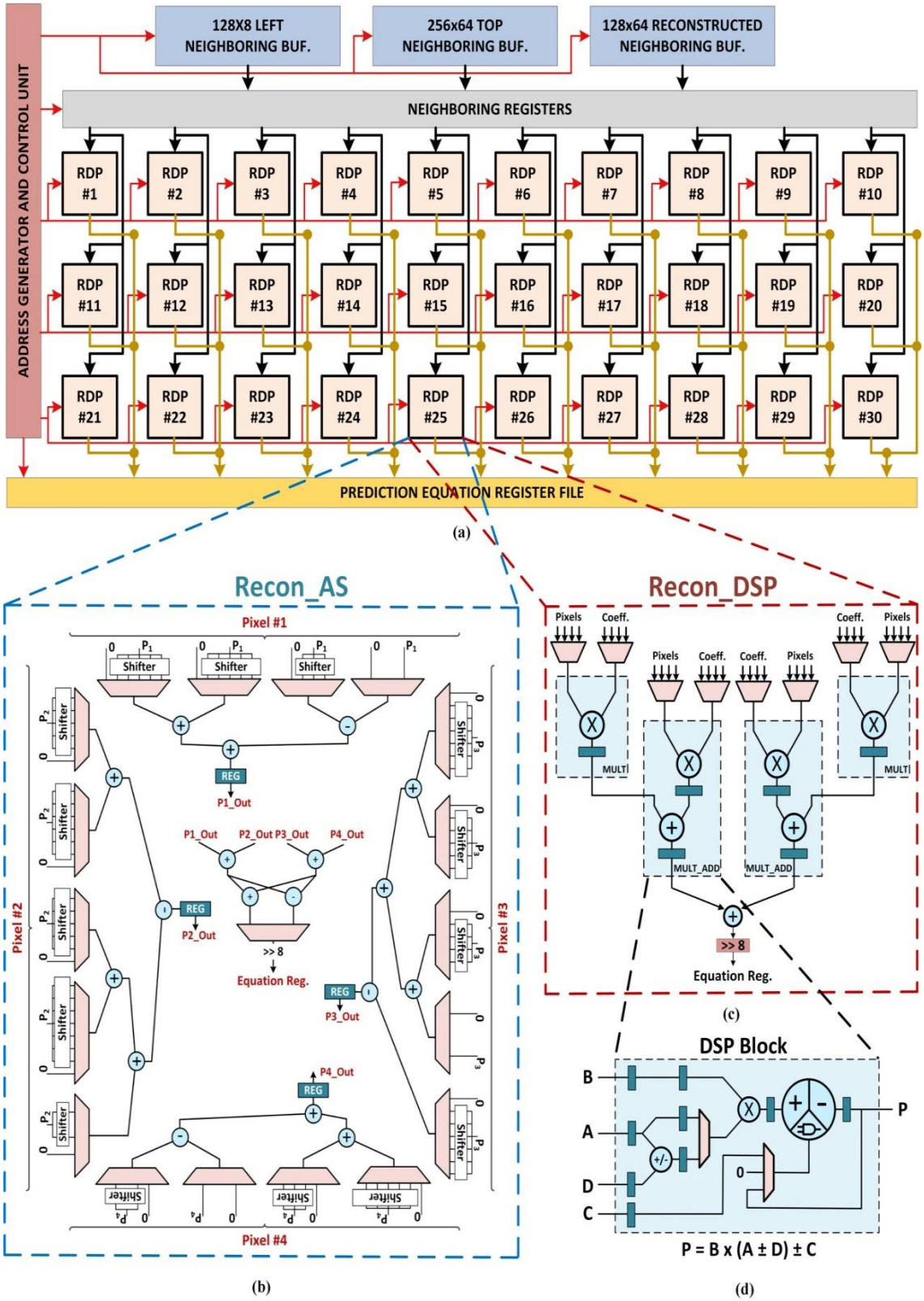

Figure 3.3 (a) VVC Reconfigurable Intra Prediction Hardware

(b) RECON_AS Datapath

(c) RECON_DSP Datapath (d) DSP Block

In order to avoid storing all 528 neighboring pixels of a 32x32 CU, it is divided into 8x8 blocks and prediction equations using pixels from the same 8x8 block are grouped together. Neighboring pixels of current 8x8 block and four 4x4 blocks are stored in 42 registers. In addition, there are extra registers to store pixels from previous 8x8 blocks, and these registers are used when pixels from two different blocks are required in a prediction equation. First, these registers are loaded. Then, thirty RDPs calculate prediction equations for current 8x8 block and four 4x4 blocks.

Two different reconfigurable datapaths are proposed. The first reconfigurable datapath (RECON_AS) is shown in Figure 3.3 (b). It implements VVC intra angular prediction equations using adders and shifters. It takes four neighboring pixels and a selection signal as input and calculates the 4-tap cubic or gaussian filter corresponding to the selection signal. It is configured by the selection signal to multiply four input pixels with coefficients of the corresponding 4-tap cubic or gaussian filter. Then, the multiplication results are added using an adder tree and the result is shifted right by eight.

FPGAs have built-in full-custom DSP blocks which can perform constant multiplications faster and with less energy than adders and shifters. A DSP block can be used to perform different constant multiplications by providing proper constant value to its input. Therefore, it is more efficient to implement constant multiplications using DSP blocks instead of using adders and shifters in an FPGA implementation. DSP block architecture is shown in Figure 3.3 (d). It has one pre-adder, one multiplier and one arithmetic logic unit (ALU). It also has optional pipeline registers.

Therefore, the second reconfigurable datapath (RECON_DSP) uses DSP blocks in FPGA to implement multiplications with constants as shown in Figure 3.3 (c). It takes four neighboring pixels and a selection signal as input and calculates the 4-tap cubic or gaussian filter corresponding to the selection signal. It multiplies four input pixels with coefficients of the corresponding 4-tap cubic or gaussian filter using four DSP blocks. Two DSP blocks which are shown as MULT in Figure 3.3 (c), multiply two input pixels with the corresponding coefficients and write the results to output registers. Other two DSP blocks, which are shown as MULT_ADD in Figure 3.3 (c), multiply the other two input pixels with the corresponding coefficients and add the multiplication results. Then, two MULT_ADD results are added and the result is shifted right by eight.

The proposed RECON_AS and RECON_DSP hardware are implemented with Verilog HDL. The Verilog codes are synthesized, placed and routed to a 28 nm FPGA. Functional simulation results and post place and route timing simulation results matched

results of VVC JEM software encoder [37]. FPGA implementations are also verified on an FPGA board as shown in Figure 3.4. The FPGA board has a 28 nm FPGA, 1 GB DRAM and several interfaces such as HDMI. The VVC intra prediction hardware and microprocessor communicates using a bus.

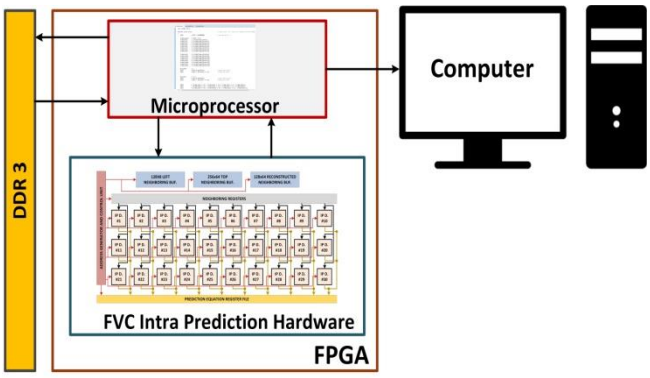

Figure 3.4 FPGA Implementation

The Verilog codes are synthesized, placed and routed to a 90 nm standard cell library as well. Since DSP blocks are only available in FPGAs, RECON_DSP ASIC implementation uses multipliers. FPGA and ASIC implementation results are given in Table 3.4.

Table 3.4 Implementation Results

	RECON_AS		**RECON_DSP**	
Technology	28 nm FPGA	90 nm ASIC	28 nm FPGA	90 nm ASIC
Slice/Gate Count	20352	96.1 K	13666	92.3 K
DFF	6237	---	4076	---
LUT	49556	---	32499	---
Memory	3.2 KB	3.2 KB	3.2 KB	3.2 KB
DSP Block	---	---	120	---
Max Freq. (MHz)	108	218	105	208
Frames per Sec.	30 1920x1080	62 1920x1080	30 1920x1080	59 1920x1080
Power (mW)	1037.8	42.2	637.7	48.4
PU Size	4,8,16,32	4,8,16,32	4,8,16,32	4,8,16,32

RECON_AS FPGA implementation uses 49556 LUTs, 6237 DFFs, 4 BRAMs. It works at 108 MHz. RECON_DSP FPGA implementation uses 32499 LUTs, 4076 DFFs,

4 BRAMs, 120 DSP blocks. It works at 105 MHz. Both FPGA implementations process 30 full HD (1920x1080) video frames per second (fps).

RECON_AS ASIC implementation uses 96.1K gates based on NAND (2x1) gate area. It works at 218 MHz. It processes 62 full HD video fps. RECON_DSP ASIC implementation uses 92.3K gates based on NAND (2x1) gate area. It works at 208 MHz. It processes 59 full HD video fps.

Power consumptions of RECON_AS and RECON_DSP FPGA implementations are estimated for Tennis, Kimono, ParkScene and Basketball Drive (1920x1080) videos at 100 MHz [35] using a gate level power estimation tool. Signal activities captured during post place and route timing simulations are used to estimate power consumptions. The energy consumptions for one frame of each video are given in Figure 3.5. Since RECON_DSP FPGA implementation uses DSP blocks instead of adders and shifters, it has up to 30.2% less energy consumption than RECON_AS FPGA implementation.

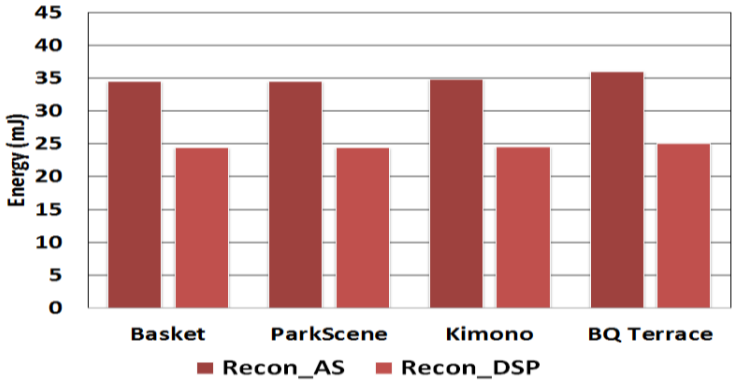

Figure 3.5 Power Consumptions

Table 3.5 Hardware Comparison

	[36]	[24]	[26]	[28]	[18]	RECON_AS	RECON_DSP
FPGA Technology	40 nm	65 nm	40 nm	40 nm	40 nm	28 nm	28 nm
DFF	849	5.5 K	110 K	---	2006	6234	4076
LUT	2381	14 K	170 K	24 K	6013	49556	32499
BRAM	3.2 KB	6 KB	---	6 KB	3.2 KB	3.2 KB	3.2 KB
Max Freq. (MHz)	150	110	219	100	166	108	105
Frames per Sec.	30	30	24	60	40	30	30
	1920x1080	3840x2160	3840x2160	1920x1080	1920x1080	1920x1080	1920x1080
PU Size	4, 8	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32

In Table 3.5, RECON_AS and RECON_DSP hardware are compared with HEVC intra prediction hardware in the literature [18, 24, 26, 28, 36]. Since VVC intra prediction

algorithm is more complex than HEVC intra prediction algorithm, RECON_AS and RECON_DSP hardware are slower and have larger area than the HEVC intra prediction hardware.

3.3 DSP Block Based FPGA Implementation of VVC Intra Prediction

An efficient FPGA implementation of VVC intra prediction for angular prediction modes of 4x4, 8x8, 16x16 and 32x32 PU sizes is proposed. The proposed FPGA implementation uses 30 identical DSP datapaths (DDP). In the proposed FPGA implementation, intra angular prediction equations are manipulated in such a way that one intra angular prediction equation is implemented using two DSP blocks and two adders. Therefore, each DDP has two DSP blocks and two adders, and it can calculate any 4-tap gaussian and cubic filter used in VVC intra angular prediction in one clock cycle by changing DSP inputs.

The proposed VVC intra angular prediction hardware is implemented using Verilog HDL. The Verilog RTL code is verified to work at 119 MHz on a Xilinx Virtex7 FPGA. The proposed VVC intra angular prediction hardware, in the worst case, can process 34 full HD (1920x1080) frames per second.

Two VVC intra prediction hardware implementations are proposed in [20]. Several HEVC intra prediction hardware implementations are proposed in the literature [18, 24, 26, 28]. The proposed VVC intra prediction hardware is compared with VVC and HEVC intra prediction hardware in the literature.

In VVC, identical prediction equations are used in an intra angular prediction mode or in different intra angular prediction modes or in the intra angular prediction modes of different PU sizes. In the proposed hardware, data reuse technique is used to calculate identical prediction equations only once. There are 4x4 (PU size) x 65 (intra angular prediction modes) = 1040 intra angular prediction equations for 4x4 PU size. Numbers of prediction equations for other PU sizes are shown in Table 3.6. The number of prediction equations calculated for 4x4 PU size is reduced to 405 by using data reuse technique. Numbers of prediction equation reductions for other PU sizes are shown in Table 3.6.

Cubic filters are used for 4x4 and 8x8 PU sizes. Total number of cubic filter prediction equations for sixty-four 4x4 PUs and sixteen 8x8 PUs in a 32x32 CU without data reuse is 133120. Gaussian filters are used for 16x16 and 32x32 PU sizes. Total number of gaussian filter prediction equations for four 16x16 PUs and one 32x32 PU in a 32x32 CU without data reuse is 133120. The numbers of cubic filter prediction

Table 3.6 Intra Angular Prediction Equation Reductions by Data Reuse

	Cubic Filters			Gaussian Filters		
	4x4 PU	**8x8 PU**	**32x32 CU**	**16x16 PU**	**32x32 PU**	**32x32 CU**
# of Pred.Equations	1040	4160	133120	16680	66560	133120
# of Pred. Equations with Data Reuse	405	1042	29478	2597	6641	11810
Reduction (%)	61.06	74.95	77.85	84.43	90.02	91.13

equations and gaussian filter prediction equations calculated are reduced by 77.85% and 91.13%, respectively with data reuse technique.

The proposed VVC intra prediction hardware is shown in Figure 3.6. It implements 65 angular prediction modes for PU sizes from 4x4 to 32x32. It has thirty parallel reconfigurable DSP datapaths (DDP). One DDP, which can be configured to implement any of the 34 cubic and gaussian filters, is shown in Figure 3.7.

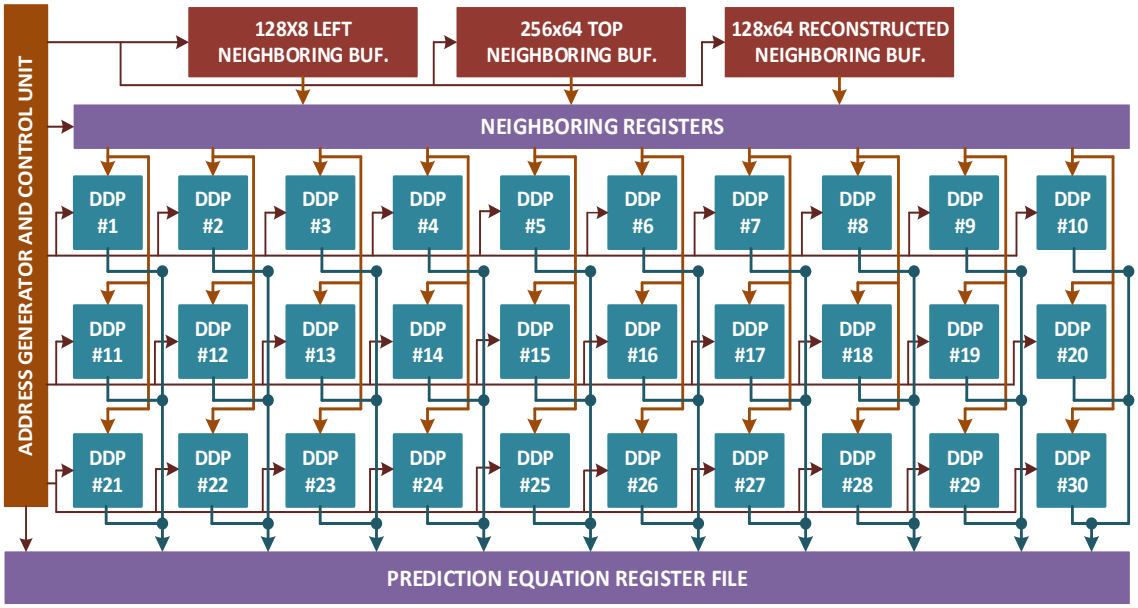

Figure 3.6 Proposed FPGA Implementation of VVC Intra Prediction

32x32 coding unit (CU) is divided into 8x8 blocks and the neighboring pixels for the current 8x8 block and four 4x4 blocks within the current 8x8 block are loaded to registers. There are extra registers to store pixels from previous blocks, in case that an equation requires pixels from different 8x8 blocks. Therefore, the number of registers to store is decreased by storing only the neighboring pixels of 8x8 blocks, instead of keeping all neighboring pixels of 32x32 CU.

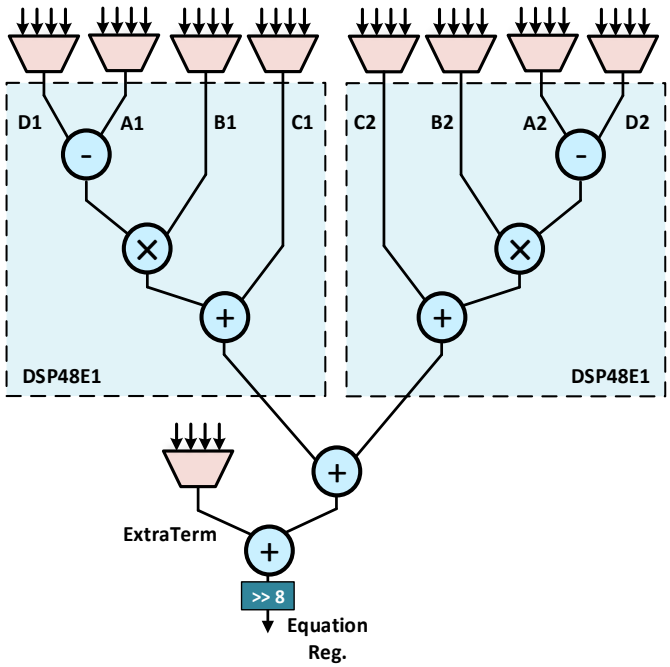

Figure 3.7 Proposed FPGA Reconfigurable DSP Datapath (DDP)

FPGAs have built-in full-custom DSP blocks which can perform constant multiplications faster and with less energy than adders and shifters. A DSP block can be used to perform different constant multiplications by providing proper constant values to its inputs. Therefore, it is more efficient to implement constant multiplications using DSP blocks instead of using adders and shifters in an FPGA implementation.

Xilinx DSP block architecture is shown in Figure 3.8. It has one pre-adder, one multiplier and one arithmetic logic unit (ALU). It also has optional pipeline registers. A DSP block can be configured to implement different operations.

In VVC, each intra angular prediction equation requires four multiplication operations to multiply four pixels with corresponding filter coefficients and three addition operations to add the results of these four multiplications. Therefore, four DSP blocks are necessary for implementing an intra angular prediction equation in its original form as in [20].

In the proposed FPGA implementation, intra angular prediction equations are manipulated in such a way that one intra angular prediction equation is implemented using two DSP blocks and two adders. Therefore, each DDP has two DSP blocks and two adders, and it can calculate any 4-tap gaussian and cubic filter used in VVC intra angular prediction in one clock cycle by changing A, B, C and D inputs of DSP blocks.

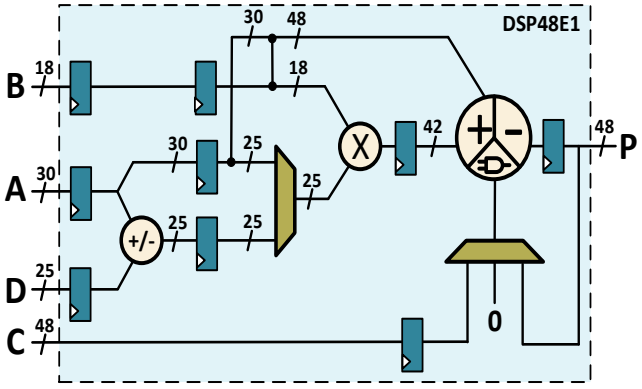

Figure 3.8 Xilinx DSP48E1 Block

In the proposed FPGA implementation, DSP blocks are configured to implement equation (3.4). Each DDP implements equation (3.5).

$$P = B * (D - A) + C \tag{3.4}$$

$$P = (B1 * (D1 - A1) + C1) + \ (B2 * (D2 - A2) + C2) + ExtraTerm \tag{3.5}$$

Four filter coefficients used in 34 VVC intra angular prediction equations are shown in Table 3.7. The A, B, C, D inputs of two DSP blocks and the extra term necessary for calculating each intra angular prediction equation using a DDP are also shown in Table 3.7. The inputs of DSP blocks are shown in the order they appear in equation (3.5). Constant numbers are given to B inputs of two DSP blocks. Pixels or shifted pixels are given to D, A and C inputs of DSP blocks. Multiplexers are used to select the proper inputs for implementing each intra angular prediction equation.

For example, the intra angular prediction equation "Filter 1" shown in Table 3.7 is implemented using a DDP as shown in equations (3.6a), (3.6b) and (3.6c).

$$P = (4 * (2 * p3 - p2) + 0) + \ \big(3 * (0 - p1) + (-p4)\big) + 256 * p2 \tag{3.6a}$$

$$P = (8 * p3 - 4 * p2 - 3 * p1 - p4 + 256 * p2) \tag{3.6b}$$

$$P = (-3 * p1 + 252 * p2 + 8 * p3 - p4) \tag{3.6c}$$

The proposed VVC intra prediction hardware is implemented using Verilog HDL. The Verilog RTL code is synthesized, placed and routed to a Xilinx XC7VX485T FFG1157 FPGA with speed grade 3 using Xilinx Vivado2017.2. The FPGA implementation is verified with post place and route simulations. The proposed

Table 3.7 DDP Configurations

Filters	Filter Coefficients				DSP Block 1				DSP Block 2				Extra Term
	Coeff1	Coeff2	Coeff3	Coeff4	B1	D1	A1	C1	B2	D2	A2	C2	
0	0	256	0	0	0	0	p2	0	0	0	0	0	p2≪8
1	-3	252	8	-1	4	p3≪1	p2	0	3	0	p1	(-p4)	p2≪8
2	-5	247	17	-3	9	p3	p2	p3≪3	-3	p1	(-p4)	(-p1)≪1	p2≪8
3	-7	242	25	-4	14	0	p2	(-p4)≪2	7	(-p1)	p3	p3≪5	p2≪8
4	-9	236	34	-5	5	(-p4)	p2≪2	0	9	(-p1)	(-p3)≪2	(-p3)≪1	p2≪8
5	-10	230	43	-7	-43	(-p3)	p2≪1	(-p2)≪4	10	p2≪4	p1	(-p4)≪3	p4
6	-12	224	52	-8	32	p3	p2	p1≪3	20	p3	p1	(-p4)≪3	p2≪8
7	-13	217	61	-9	-9	p4	p2	(-p3)≪2	13	(-p1)	(-p2)≪4	p3≪6	p3
8	-14	210	70	-10	-12	p1	p2	(-p4)≪1	70	p2	-p3	(-p4)≪3	p2≪7
9	-15	203	79	-11	-75	(-p3)	p2	p3≪2	15	(-p1)	p4	p4≪2	p2≪7
10	-16	195	89	-12	61	p3	p2	(-p1)≪4	12	p3	p4	p3≪4	p2≪8
11	-16	187	98	-13	69	p3	p2	(-p1)≪4	13	p3	p4	p3≪4	p2≪8
12	-16	179	10	-14	-51	(-p3)≪1	p2	(-p1)≪4	5	p3	p4≪1	(-p4)≪2	p2≪7
13	-16	170	11	-14	86	p3	p2	(-p1)≪4	14	p3	p4	p3≪4	p2≪8
14	-17	162	12	-15	-17	p1	p2≪1	p4	126	p3	0	(-p4)≪4	p2≪7
15	-16	153	13	-16	103	p3	p2	(-p1)≪4	16	p3	p4	p3≪4	p2≪8
16	-16	144	14	-16	-144	(-p3)	p2	0	16	(-p1)	p4	0	0
17	47	161	47	1	-161	0	p2	p4	47	p1	(-p3)	0	0
18	43	161	51	1	-161	0	p2	p4	43	p1	(-p3)	p3≪3	0
19	40	160	54	2	-32	(-p1)	p2	p4≪1	54	0	(-p3)	p1≪3	p2≪7
20	37	159	58	2	-159	0	p2	p4≪1	37	p1	(-p3)≪1	(-p3)≪4	0
21	34	158	62	2	-62	(-p3)	p2≪1	0	34	p1	(-p2)	p4≪1	0
22	31	156	67	2	-5	(-p3)	p2≪2	p2≪3	31	p1	(-p3)≪1	p4≪1	p2≪7
23	28	154	71	3	-71	(-p3)	p2≪1	p1≪5	3	p4	(-p2)≪2	(-p1)≪2	0
24	26	151	76	3	-76	(-p3)	p2≪1	(-p2)	3	p1≪3	(-p4)	p1≪1	0
25	23	149	80	4	-21	(-p1)	p2	p1≪1	80	p3	0	p4≪2	p2≪7
26	21	146	85	4	-18	0	p2	p4≪2	21	p1	(-p3)≪2	p3	p2≪7
27	19	142	90	5	-14	(-p3)	p2	p4≪2	19	p1	(-p3)≪2	p4	p2≪7
28	17	139	94	6	-11	(-p3)≪3	p2	p1≪4	6	p3	(-p4)	p1	p2≪7
29	16	135	99	6	-7	(-p3)≪4	p2	p1≪4	6	p4	p3≪1	(-p3)	p2≪7
30	14	131	10	7	-3	(-p3)≪5	p2	p3≪3	7	p1≪1	(-p4)	0	p2≪7
31	13	127	10	8	-127	0	p2	p4≪3	13	p1	(-p3)≪3	p3≪2	0
32	11	123	11	9	-113	(-p3)	p2	p1	10	p1	(-p2)	p4≪3	p4
33	10	118	11	10	-118	(-p3)	p2	0	0	p1	(-p4)	0	0

FPGA implementation uses 5766 DFFs, 46382 LUTs, 4 BRAMs and 60 DSP48E1s blocks. It works at 119 MHz. It can process 34 full HD (1920x1080) video frames per second (fps).

The proposed VVC intra prediction hardware is compared with HEVC and VVC intra prediction hardware in the literature in Table 3.8. Since VVC intra prediction algorithm is more complex than HEVC intra prediction algorithm, the proposed VVC intra prediction hardware implementation and the two VVC intra prediction hardware implementations proposed in [20] are slower and have more area than the HEVC intra prediction hardware implementations [18, 24, 26, 28].

Table 3.8 Hardware Comparison

	[10]	[12]	[13]	[14]	[15]	[11] RECON_AS	[11] RECON_DSP	Proposed
FPGA	Xilinx 6	Stratix III	Arria II	Virtex 6	Xilinx 6	Virtex 7	Virtex 7	Virtex 7
FPGA Technology	40 nm	65 nm	40 nm	40 nm	40 nm	28 nm	28 nm	28 nm
Standard	HEVC	HEVC	HEVC	HEVC	HEVC	VVC	VVC	VVC
DFF	849	5.5 K	110 K	---	2006	6234	4076	5766
LUT	2381	14 K	170 K	24 K	6013	49556	32499	46382
BRAM	3.2 KB	6 KB	---	6 KB	3.2 KB	3.2 KB	3.2 KB	3.2 KB
DSP Block	---	---	---	---	---	---	120	60
Max Freq. (MHz)	150	110	219	100	166	108	105	119
Frames per Sec.	30	30	24	60	40	30	30	34
	1920x1080	3840x2160	3840x2160	1920x1080	1920x1080	1920x1080	1920x1080	1920x1080
PU Size	4, 8	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32

RECON_AS hardware implements VVC intra prediction using adders and shifters [20]. It does not use DSP blocks. RECON_DSP hardware implements VVC intra prediction using DSP blocks [20]. It uses four DSP blocks and one adder for implementing an intra angular prediction equation. The proposed VVC intra prediction hardware is faster than both RECON_AS and RECON_DSP hardware. It uses 50% less DSP blocks than RECON_DSP hardware.

CHAPTER IV

VVC FRACTIONAL INTERPOLATION HARDWARE

HEVC standard uses 3 different 8-tap FIR filters for fractional interpolations and provides 1/4 fractional pixel accuracy. However, VVC standard uses 15 different 8-tap FIR filters for fractional interpolations and provides 1/16 fractional pixel accuracy. Therefore, VVC fractional interpolation has much higher computational complexity than HEVC fractional interpolation.

In this thesis, a reconfigurable VVC fractional interpolation hardware for motion compensation (MC) is proposed. The proposed hardware supports all prediction unit (PU) sizes. It interpolates necessary fractional pixels for the fractional pixel location in an 8x8 PU pointed by the given fractional pixel accurate motion vector. For larger PU sizes, the PU is divided into 8x8 blocks, and the blocks are interpolated separately. Since the proposed hardware is used for motion compensation stage of VVC encoder and decoder, only one fractional pixel per integer pixel is required. Therefore, the proposed hardware has a reconfigurable datapath which can be configured to implement any of the 15 different 8-tap FIR filters.

The proposed VVC fractional interpolation hardware is implemented using Verilog HDL. The Verilog RTL code is verified to work at 250 MHz on a Xilinx Virtex 7 FPGA. The proposed VVC fractional interpolation hardware, in the worst case, can process 66 quad full HD (3840x2160) frames per second. The proposed reconfigurability reduced the power consumption of FPGA implementation of the proposed VVC fractional interpolation hardware by 77%.

The proposed hardware is the first VVC fractional interpolation hardware for motion compensation in the literature. Several HEVC fractional interpolation hardware implementations are proposed in the literature [38]-[43]. The proposed VVC fractional interpolation hardware is compared with them.

4.1 VVC Fractional Interpolation Algorithm

VVC standard uses 15 different 8-tap FIR filters for fractional pixel interpolation. The coefficients of these 15 FIR filters are shown in Table 4.1. A_{-3} – A_4 show input pixels for a filter where sub-indices represent the indices of coefficients. The F_7 8-tap FIR filter equation is shown in equation (4.1) as an example.

Table 4.1 VVC Fractional Interpolation Filters

Filters	Coefficients							
	A_{-3}	A_{-2}	A_{-1}	A_0	A_1	A_2	A_3	A_4
1	0	1	-3	63	4	-2	1	0
2	-1	2	-5	62	8	-3	1	0
3	-1	3	-8	60	13	-4	1	0
4	-1	4	-10	58	17	-5	1	0
5	-1	4	-11	52	26	-8	3	-1
6	-1	3	-9	47	31	-10	4	-1
7	-1	4	-11	45	34	-10	4	-1
8	-1	4	-11	40	40	-11	4	-1
9	-1	4	-10	34	45	-11	4	-1
10	-1	4	-10	31	47	-9	3	-1
11	-1	3	-8	26	52	-11	4	-1
12	0	1	-5	17	58	-10	4	-1
13	0	1	-4	13	60	-8	3	-1
14	0	1	-3	8	62	-5	2	-1
15	0	1	-2	4	63	-3	1	0

$$F_7 = (-A_{-3} + 4*A_{-2} - 11*A_{-1} + 45*A_0 + 34*A_1 - 10*A_2 + 4*A_3 + 4*A_4) >> 6 \quad (4.1)$$

Integer pixels, fractional pixels and FIR filters used to interpolate these fractional pixels are shown in Figure 4.1. There are 255 fractional (half and quarter) pixels for one integer pixel. There are 15 half-pixels between two neighboring horizontal integer pixels called horizontal half-pixels. There are 15 half-pixels between two neighboring vertical integer pixels called vertical half-pixels. These 15 horizontal and 15 vertical half-pixels are interpolated from nearest integer pixels in horizontal and vertical directions, respectively, using 15 different 8-tap FIR filters. There are 15x15=225 quarter-pixels between 15 horizontal and 15 vertical half-pixels. These quarter-pixels are interpolated from nearest horizontal half-pixels using 15 different 8-tap FIR filters.

VVC fractional interpolation algorithm used for motion compensation interpolates necessary fractional pixels for one out of 255 fractional pixel locations pointed by the given 1/16 pixel accurate motion vector. Necessary fractional pixels are determined using x fraction and y fraction of the given 1/16 pixel accurate motion vector. If either x fraction or y fraction is zero, only necessary half-pixels are interpolated. If neither x fraction nor y fraction is zero, horizontal half-pixels necessary to interpolate the quarter-pixels are

interpolated first. Then, the necessary quarter-pixels are interpolated using these horizontal half-pixels.

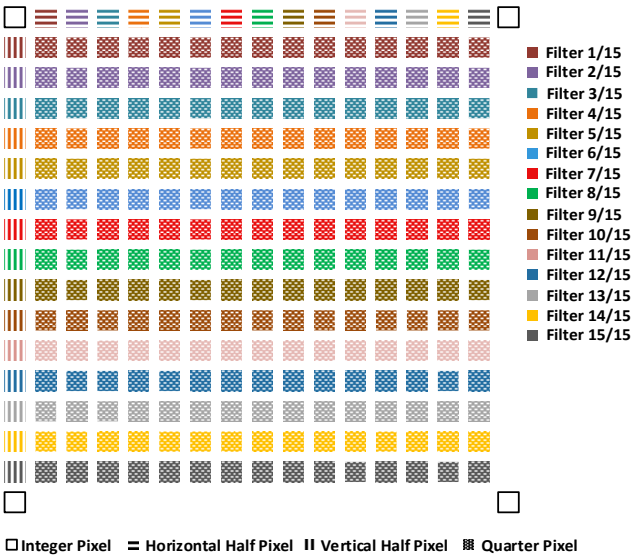

Figure 4.1 Integer, Half and Quarter Pixels

4.2 Proposed VVC Fractional Interpolation Hardware

The proposed reconfigurable VVC fractional interpolation hardware for all PU sizes is shown in Figure 4.2. The proposed hardware interpolates the necessary fractional pixels for luma component of an 8x8 PU for a given 1/16 pixel accurate motion vector using integer or half-pixels. For larger PU sizes, the PU is divided into 8x8 blocks and these blocks are interpolated separately. For example, a 16x16 PU is divided into four 8x8 blocks and each 8x8 block is interpolated separately.

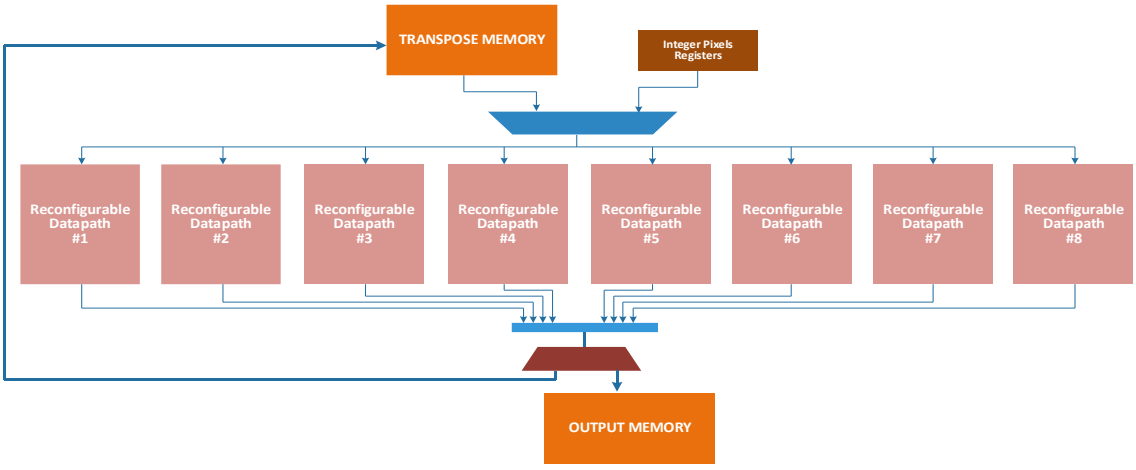

Figure 4.2 Proposed VVC Fractional Interpolation Hardware

Since 15x8 horizontal half-pixels are necessary for interpolating quarter-pixels, 15x8x8 on-chip transpose memory is used to store horizontal half-pixels necessary for interpolating quarter-pixels in certain cases. The horizontal half-pixels interpolated from nearest integer pixels in horizontal direction are stored in transpose memory horizontally in 15 clock cycles. Then, 15 horizontal half-pixels are read vertically from transpose memory in each clock cycle to interpolate quarter-pixels.

The proposed hardware takes 15 integer pixels in each clock cycle. It interpolates 8 fractional pixels in each clock cycle using 8 parallel reconfigurable datapaths. If the necessary fractional pixels are half-pixels, 8x8 half-pixels are interpolated using the integer pixels in 8 clock cycles. If the necessary fractional pixels are quarter-pixels, 15x8 horizontal half-pixels are interpolated using the integer pixels in 15 clock cycles. Then, 8x8 quarter-pixels are interpolated using these horizontal half-pixels in 8 clock cycles. There are three pipeline stages in the proposed hardware. Therefore, the proposed hardware interpolates the half-pixels and quarter-pixels for an 8x8 PU in 11 and 29 clock cycles, respectively.

15 different 8-tap FIR filters are used to interpolate half-pixels and quarter-pixels. Last 7 FIR filters are symmetric of the first 7 FIR filters. Therefore, in this thesis, a reconfigurable datapath which implements the first 8 FIR filters is proposed. It can be configured to calculate output of any of the first 8 FIR filters. To calculate output of one of the last 7 FIR filters using the proposed reconfigurable datapath, inputs are reversed, and corresponding symmetric filter is selected.

The proposed reconfigurable datapath is shown in Figure 4.3. It implements multiplications with constant coefficients using adders and shifters. It has 14 adders/subtractors and their inputs are determined by a filter selection signal. It selects different input pixels with different shift amounts for each fractional interpolation equation using input multiplexers as shown in Table 4.2.

In this thesis, a baseline VVC fractional interpolation hardware is also designed and implemented for comparison. The baseline hardware has the same architecture as the proposed hardware. The only difference is their datapaths. In the baseline hardware datapath, all 15 FIR filters are implemented separately and output of one FIR filter is selected based on filter selection signal. Therefore, the baseline hardware datapath has 91 adders while the proposed reconfigurable datapath has 14 adders.

The proposed and the baseline VVC fractional interpolation hardware are implemented using Verilog HDL. The Verilog RTL codes are verified with RTL

simulations. The Verilog RTL codes are synthesized and mapped to a Xilinx VC7VX330T-3FFG1157 FPGA using Xilinx ISE 14.7. The FPGA implementations are verified with post place and route simulations. The simulation results matched the results of a software implementation of VVC fractional interpolation algorithm.

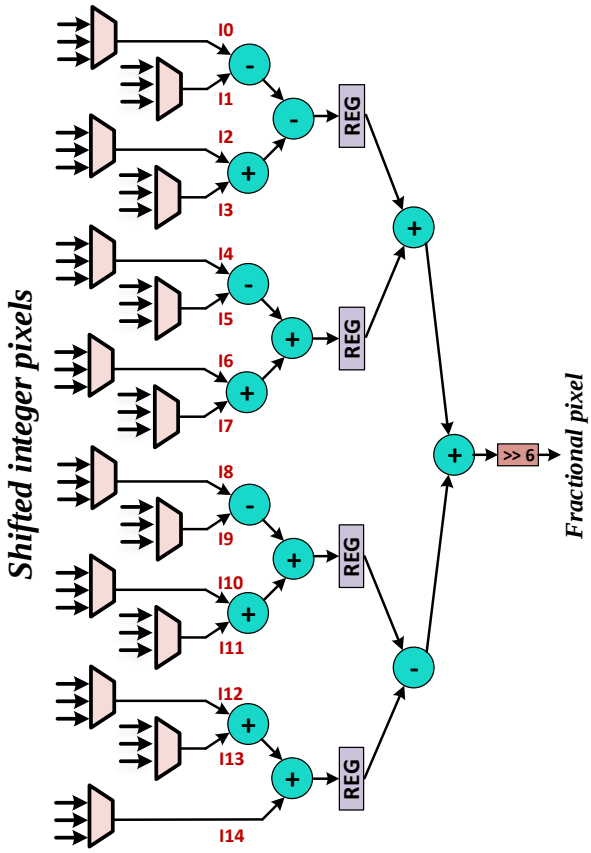

Figure 4.3 Proposed Reconfigurable Datapath

Table 4.2 Reconfigurable Datapath Inputs

Filters	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14
1	0	0	B	0	C<<2	D<<6	C	D	E<<2	0	0	F<<1	0	G	0
2	A	B<<1	0	C<<2	C<<1	D<<6	C	D<<1	E<<3	0	0	F<<1	F	G	0
3	A	B<<1	B	C<<3	0	D<<6	0	D<<2	E<<4	E	E<<2	F<<2	0	G	0
4	A	B<<2	0	C<<3	C<<1	D<<6	D<<1	D<<3	E<<4	E	0	F<<2	F	G	0
5	A	B<<2	C	C<<3	C<<2	D<<6	D<<2	D<<4	E<<5	E<<1	E<<3	F<<3	G	G<<2	H
6	A	B<<1	B	C<<3	C<<1	D<<5	D<<4	D	E<<5	C	E	F<<3	F<<1	G<<2	H
7	A	B<<2	D	C<<3	C<<1	D<<5	D<<4	D<<2	E<<5	E<<1	C	F<<3	F<<1	G<<2	H
8	A	B<<2	F	C<<3	C<<1	D<<5	D<<3	0	E<<5	E<<3	C	F<<3	F<<2	G<<2	H

As shown in Figure 4.4, FPGA implementations are also verified to work correctly on a Xilinx Virtex 7 VC707 FPGA board which includes an FPGA, 1 GB DRAM and interfaces such as UART and HDMI. Microblaze processor reads video frames from

computer, stores them to DDR memory and sends them to FPGA using high-speed AXI-4 bus. The proposed hardware interpolates the video frames. Then, interpolated video frames are displayed on HDMI monitor.

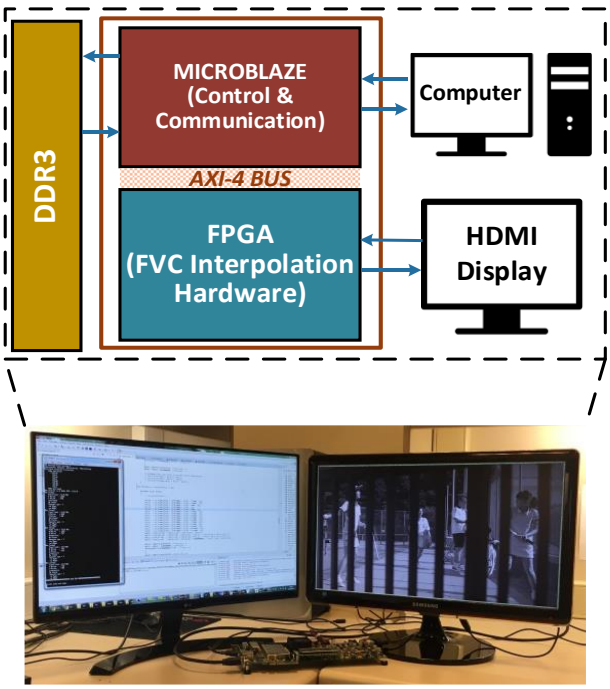

Figure 4.4 FPGA Board Implementation

As shown in Table 4.2, FPGA implementation of the proposed VVC fractional interpolation hardware uses 1688 DFFs and 4467 LUTs. It can work at 250 MHz, and it can process 66 quad full HD (3840x2160) frames per second. FPGA implementation of the baseline VVC fractional interpolation hardware uses 5446 DFFs and 12016 LUTs. It can work at 238 MHz, and it can process 63 quad full HD (3840x2160) frames per second.

Table 4.3 Implementation Results

	Baseline		**Proposed**	
Technology	Xilinx Virtex 7	TSMC 90 nm	Xilinx Virtex 7	TSMC 90 nm
Slice/Gate Count	3630	48.3 K	1407	11.7 K
DFF	5446	---	1688	---
LUT	12016	---	4467	---
Max. Freq. (MHz)	238	417	250	357
Frames per Second	63 (3840x2160)	110 (3840x2160)	66 (3840x2160)	95 (3840x2160)

The Verilog RTL codes of the baseline and proposed VVC fractional interpolation hardware are also synthesized to TSMC 90 nm standard cell library, and the resulting

netlists are placed and routed. As shown in Table 4.3, ASIC implementations of the baseline and proposed hardware use 48.3K and 11.7K gates, respectively, based on NAND (2x1) gate area excluding on-chip memory. ASIC implementations of the baseline and proposed hardware can work at 417 and 357 MHz, respectively, and they can process 110 and 95 quad full HD frames per second, respectively.

Since the proposed hardware is the first VVC fractional interpolation hardware for motion compensation in the literature, it is compared with HEVC fractional interpolation hardware in the literature [38]-[43]. The comparison is shown in Table 4.4. The HEVC fractional interpolation hardware proposed in [38] is designed for motion compensation. The others can be used for both motion estimation (ME) and motion compensation.

Table 4.4 Hardware Comparison

	[38]	[39]	[40]	[41]	[42]	[43]	Proposed
FPGA	Xilinx Virtex 6	Xilinx Virtex 6	Arria II GX	Xilinx Virtex 5	Stratix III	Xilinx Virtex 6	Xilinx Virtex 7
Slices	---	---	---	2181	---	1498	1407
LUTs	3005	3929	18831	5017	7701	3806	4467
Block RAMs	2	6	---	2	---	---	---
Max. Freq. (MHz)	100	200	200	283	278	233	250
Frames per Second	64 2560x1600	30 3840x2160	60 1920x1080	30 2560x1600	60 3840x2160	35 3840x2160	66 3840x2160
Design	Only MC	ME + MC	ME + MC	ME + MC	ME + MC	ME + MC	Only MC
Standard	HEVC	HEVC	HEVC	HEVC	HEVC	HEVC	VVC

Since VVC fractional interpolation has higher computational complexity than HEVC fractional interpolation, the proposed hardware has higher area than the HEVC fractional interpolation hardware proposed in [38]. However, since the proposed hardware is designed for motion compensation, it does not have higher area than the other HEVC fractional interpolation hardware in the literature.

Power consumptions of the baseline and proposed hardware are estimated using Xilinx XPower Analyzer tool. Post place and route timing simulations are performed for Tennis and Kimono (1920x1080) video frames at 100 MHz [35]. The signal activities of these timing simulations are stored in VCD files, and they are used for estimating the power consumptions of FPGA implementations. The power consumptions of both the baseline and proposed hardware are shown in Table 4.5. Clock, signal and logic power consumptions are given for detailed analysis. Total power consumption of the proposed hardware for Tennis and Kimono frames is 76.21% and 77.02% less than that of the baseline hardware, respectively.

Table 4.5 Power Consumption Results

	Baseline		**Proposed**	
Frame	**Tennis**	**Kimono**	**Tennis**	**Kimono**
Clock (mW)	68.33	68.33	13.84	13.84
Signal (mW)	96.75	131.64	16.64	22.58
Logic (mW)	99.27	135.36	32.40	40.64
Total Power (mW)	264.35	335.33	62.88	77.06
Power Reduction	---	---	76.21 %	77.02 %

CHAPTER V

APPROXIMATE VIDEO COMPRESSION HARDWARE

Approximate computing is a promising solution to increased computational complexity of signal processing applications [44]-[52]. Approximate computing allows designing faster, smaller area and lower power consuming hardware than the exact optimized hardware designs, by trading off speed, area and power consumption with quality. Therefore, it can be used in error tolerant applications.

Different approximate computing approaches are proposed in the literature [53]-[57]. A commonly used approximate computing approach is using general purpose approximate arithmetic circuits such as approximate adders and multipliers [58]-[63] instead of exact arithmetic circuits. These approximate arithmetic circuits have different accuracy, speed, area and power consumption. The ones satisfying accuracy, speed, area and power consumption requirements of an application can be used for that application.

Several approximate adders are proposed in the literature [58]-[60]. Almost Correct Adder (ACA-I) proposed in [58] splits an adder into overlapping sub-adders with fixed size. Since it has shorter critical path, it is faster than exact adder. It has larger area than exact adder because of overlapping sub-adders. However, its accuracy is high.

Error Tolerant Adder (ETA-II) proposed in [59] splits input operands into accurate and inaccurate parts. Accurate part includes several most significant bits (MSB) and inaccurate part includes the remaining least significant bits (LSB). Accurate part is added exactly. Since MSBs affect error magnitude more than LSBs, this reduces error. Inaccurate part is added approximately without generating or taking in carry signal.

Generic Accuracy Configurable Adder (GeAr) proposed in [60] provides a generalized model for accuracy-configurable adders which allows adders to be configured as various approximate adders such as ACA-I and ETA-II. It also has a reconfigurable error correction unit which enables computation of accurate results when required.

Several approximate multipliers are proposed in the literature [61]-[63]. The approximate multiplier proposed in [61] first generates partial products. Then, the partial products are reduced to addition of two operands using approximate 4-2 compressors. Finally, these two partial products are added with an exact adder. In the paper, two different approximate 4-2 compressors and two different approximate multiplier architectures are proposed. Four different approximate multipliers are proposed by using these two different approximate 4-2 compressors and two different multiplier architectures. First multiplier architecture uses only approximate 4-2 compressors to reduce the partial products to two operands. Second multiplier architecture uses approximate 4-2 compressors for LSBs and exact 4-2 compressors for MSBs.

The accuracy configurable multiplier proposed in [62] divides multiplicand and multiplier into two parts named as high and low. Two parts of the multiplicand and two parts of the multiplier are multiplied separately and added. These four multiplications can be done using exact or approximate multipliers. The approximate multiplier first generates partial products. Then, addition of partial products is done approximately or exactly depending on bit position. LSBs are calculated using exact addition. Middle bits are all estimated to be 1 and a carry value is estimated. MSBs are calculated using exact addition and the estimated carry value.

The approximate multiplier proposed in [63] first generates partial products. Then, it reduces the partial products to addition of three operands using a novel method called ‘an incomplete adder cell’ (iCAC) and OR gates which have lower complexity than exact addition. These three operands are reduced to two operands using exact addition. MSBs of last two operands are added using exact addition. Middle bits are added using a carry-maskable adder (CMA). Accuracy of CMA is controlled by a mask input. LSBs are calculated using OR gates instead of using exact addition.

5.1 Novel Approximate Absolute Difference Hardware

Absolute difference (AD) operation is heavily used in many applications such as motion estimation (ME) for video compression [47], ME for frame rate conversion [48], stereo matching for depth estimation [49]. Since most of the applications using AD operation are error tolerant by their nature, approximate hardware designs can be used in these applications.

Approximate AD hardware can be designed by using general purpose approximate adders proposed in the literature in exact AD hardware. However, better approximate AD

hardware can be designed by using special approximation techniques for AD hardware instead of using general purpose approximate adders.

In this thesis, four novel approximate AD hardware designs are proposed. These approximate AD hardware designs use special approximation techniques for AD hardware instead of using general purpose approximate adders proposed in the literature. The proposed approximate AD hardware are compared with two exact baseline AD hardware and ten other approximate AD hardware.

These ten approximate AD hardware are obtained by using five approximate adders proposed in the literature [50]-[52] in the two exact baseline AD hardware. These two exact baseline AD hardware have exact subtractors. Therefore, approximate adders proposed in the literature are used as approximate subtractors by giving 2's complement of one input to the approximate adders instead of the original input.

Two exact baseline AD hardware and all fourteen approximate AD hardware are implemented using Verilog HDL. The Verilog RTL codes are synthesized and mapped to a Xilinx XC6VLX130T FF1156 FPGA with speed grade 3 using Xilinx ISE 14.7. The FPGA implementations are verified with post place and route simulations.

The proposed approximate AD hardware implementations have higher performance, smaller area and lower power consumption than exact AD hardware implementations at the expense of lower accuracy. The proposed approximate AD hardware implementations have less error, smaller area and lower power consumption than the approximate AD hardware implementations which use approximate adders proposed in the literature [50]-[52].

In the hardware implementations of applications using AD operations such as video compression, frame rate conversion and depth estimation, large number of parallel AD hardware such as 512, 1024 are used. In this thesis, area and power consumption results are reported for one AD hardware. Area and power consumption reductions achieved by using the approximate AD hardware proposed in this thesis would be much larger for the hardware implementations using large number of parallel AD hardware.

5.1.1 Proposed Approximate Absolute Difference Hardware

The three proposed approximate AD hardware are shown in Figure 5.1. As shown in Figure 5.1 (a), proposed_0 hardware consists of a subtractor and XOR gates. First, two 8-bit inputs A and B are subtracted with an exact subtractor hardware. Then, each bit of the subtraction result is XOR'ed with the sign bit of the subtraction result. If A >= B, the

sign bit is 0. Therefore, each bit is XOR'ed with 0. In this case, proposed_0 hardware computes the correct absolute difference. If A < B, the sign bit is 1. Therefore, each bit is XOR'ed with 1. In this case, the output of proposed_0 hardware is 1 less than the correct absolute difference. Therefore, the maximum error of proposed_0 hardware is 1.

As shown in Figure 5.1 (b), in proposed_1 hardware, the most significant 7 bits of subtraction result is XOR'ed with the sign bit. But, the least significant bit of the subtraction result is not XOR'ed with the sign bit. Therefore, proposed_1 hardware has 1 less XOR gate than proposed_0 hardware. However, its maximum error is 2 which is 1 more than the maximum error of proposed_0 hardware.

As shown in Figure 5.1 (c), in proposed_2 hardware, the most significant 6 bits of subtraction result is XOR'ed with the sign bit. But, the least significant 2 bits of the subtraction result is not XOR'ed with the sign bit. Therefore, proposed_2 hardware has 2 less XOR gates than proposed_0 hardware. However, its maximum error is 4 which is 3 more than the maximum error of proposed_0 hardware.

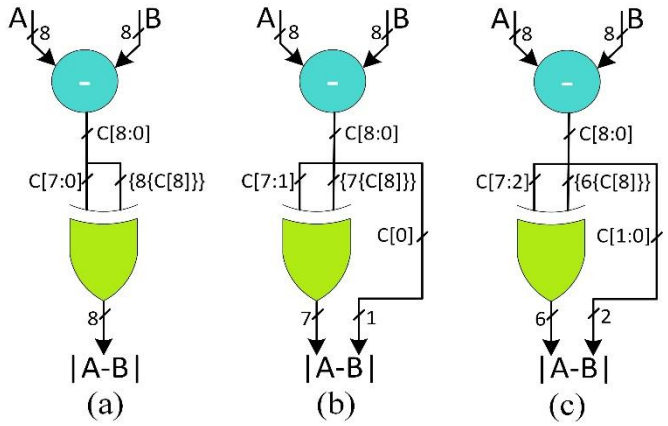

Figure 5.1 Proposed Approximate Absolute Difference Hardware
(a) proposed_0, (b) proposed_1, (c) proposed_2

The proposed_half approximate AD hardware is shown in Figure 5.2. It uses two 4-bit subtractors instead of one 8-bit subtractor. The results of two 4-bit subtractors are XOR'ed with the sign bit of first 4-bit subtraction result. The middle bit of AD is calculated by XOR'ing sign bits of both 4-bit subtraction results and the least significant bit of first 4-bit subtraction result.

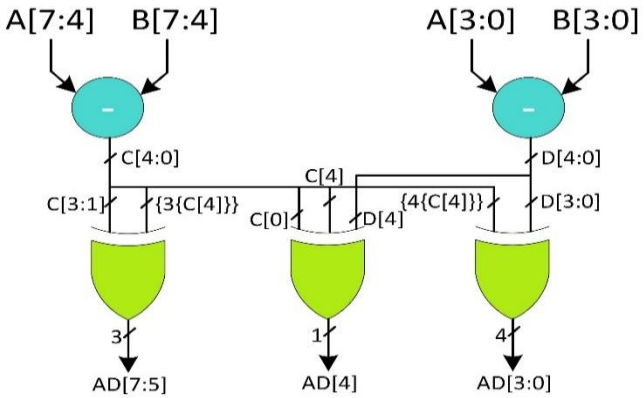

Figure 5.2 Proposed Approximate Absolute Difference Hardware (proposed_half)

Since using two 4-bit subtractors instead of one 8-bit subtractor significantly reduces the delay of critical path which is carry propogation, proposed_half hardware is faster than proposed_0, proposed_1 and proposed_2 hardware. However, proposed_half hardware has a maximum error of 33 which is larger than the maximum errors of proposed_0, proposed_1 and proposed_2 hardware.

The four approximate AD hardware proposed in this thesis are compared with ten other approximate AD hardware. These ten approximate AD hardware are obtained by using five approximate adders proposed in the literature [50]-[52] in the two exact baseline AD hardware shown in Figure 5.3. These two exact baseline AD hardware have exact subtractors. Therefore, approximate adders proposed in the literature are used as approximate subtractors by giving 2's complement of one input to the approximate adders instead of the original input.

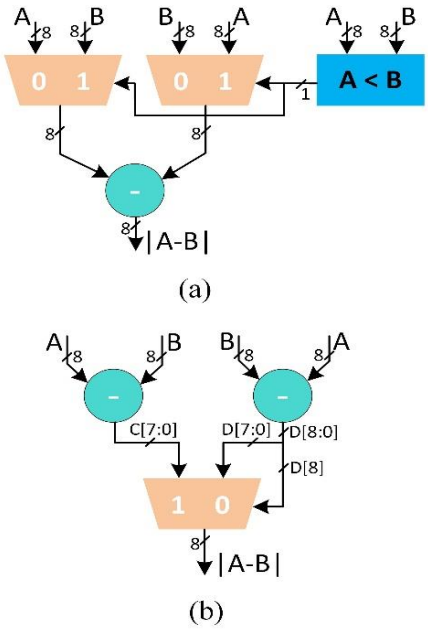

Figure 5.3 Exact Absolute Difference Hardware (a) Baseline 1 (b) Baseline 2

Ten approximate AD hardware are obtained by replacing exact subtractors in the two exact baseline AD hardware with the following five approximate adders in the literature; Almost Correct Adder I (ACA_I) [50], Almost Correct Adder II (ACA_II) [50], Error Tolerant Adder II (ETA_II) [51], Generic Accuracy Configurable Adder with N, R and P values of 8, 1 and 1, respectively (GEAR_N8_R1_P2) [52] and Generic Accuracy Configurable Adder with N, R and P values of 8, 2 and 4, respectively (GEAR_N8_R2_P4) [52].

Accuracy analysis of the approximate AD hardware proposed in this thesis and these ten approximate AD hardware is shown in Table 5.1. For example, B1_ACA_I hardware is obtained by using ACA_I approximate adder in the exact baseline 1 absolute difference hardware. B2_ACA_I hardware is obtained by using ACA_I approximate adder in the exact baseline 2 absolute difference hardware. The eight other approximate AD hardware in Table 5.1 are obtained similarly. The proposed_0, proposed_1 and proposed_2 hardware have less accuracy than the ten approximate AD hardware. However, they have much less maximum and average error than the ten approximate AD hardware.

Table 5.1 Accuracy Analysis of Approximate Absolute Difference Hardware

	Max. Error	Average Error	Accuracy (%)
Proposed_0	1	0.498	50.195
Proposed_1	2	0.496	75.195
Proposed_2	4	0.992	62.695
Proposed_half	33	7.637	39.941
B1_ACA_I	128	2.188	96.679
B2_ACA_I	128	3.418	95.312
B1_ACA_II	64	5.906	84.179
B2_ACA_II	64	7.168	81.250
B1_ETAII	64	5.926	84.179
B2_ETAII	64	7.168	81.250
B1_GeAr_R1_P2	144	10.172	75.488
B2_GeAr_R1_P2	144	14.168	69.922
B1_GeAr_R2_P4	64	1.125	98.242
B2_GeAr_R2_P4	64	1.480	97.656

5.1.2 Implementation Results

Two exact baseline AD hardware and all fourteen approximate AD hardware are implemented using Verilog HDL. The Verilog RTL codes are verified with RTL simulations. RTL simulation results matched the results of MATLAB implementations of the corresponding approximate AD algorithms.

The Verilog RTL codes are synthesized and mapped to a Xilinx XC6VLX130T FF1156 FPGA with speed grade 3 using Xilinx ISE 14.7. The FPGA implementations are verified with post place and route simulations. Post place and route simulation results matched the results of MATLAB implementations of the corresponding approximate AD algorithms.

Power consumptions of all the FPGA implementations are estimated using Xilinx XPower Analyzer tool. Post place and route timing simulations are performed at 100 MHz and the signal activities of these timing simulations are stored in VCD files. Then, they are used for estimating the power consumptions of the FPGA implementations.

Table 5.2 FPGA Implementation Results of Approximate Absolute Difference Hardware

	LUT	Slice	Frequency (MHz)	Power (mW)
Exact Baseline 1	20	15	499	4.64
Exact Baseline 2	26	10	599	4.74
Proposed_0	19	10	651	5.26
Proposed_1	17	10	653	5.26
Proposed_2	16	9	671	4.27
Proposed_half	18	7	800	4.52
B1_ACA_I	36	12	453	5.95
B2_ACA_I	34	15	624	5.59
B1_ACA_II	31	13	458	5.54
B2_ACA_II	30	15	689	5.22
B1_ETAII	31	15	457	5.62
B2_ETAII	30	17	688	5.17
B1_GeAr_R1_P2	29	13	499	5.33
B2_GeAr_R1_P2	26	19	771	5.03
B1_GeAr_R2_P4	32	17	449	5.23
B2_GeAr_R2_P4	34	14	608	5.21

The FPGA implementation results are shown in Table 5.2. All four approximate AD hardware proposed in this thesis have higher performance and less area than both exact baseline hardware. Proposed_2 and proposed_half hardware also have lower power consumption than both exact baseline hardware.

The proposed_0, proposed_1 and proposed_2 hardware have less area than the other ten approximate AD hardware. They also have much less maximum and average error than the other ten approximate AD hardware. Proposed_2 and proposed_half hardware also have lower power consumption than the other ten approximate AD hardware.

Average error vs. delay graph for all 14 approximate AD hardware is shown in Figure 5.4. Proposed_0, proposed_1 and proposed_2 hardware have the best average error vs. delay performance.

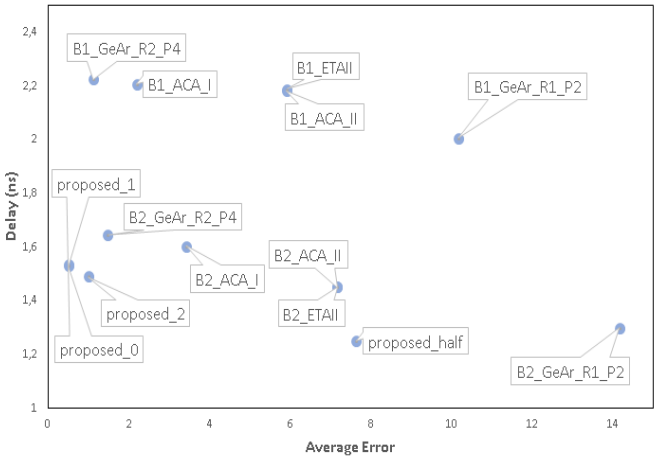

Figure 5.4 Average Error vs. Delay Graph

Proposed_0 hardware has the largest area and power consumption among the four approximate AD hardware proposed in this thesis. However, it has the smallest maximum and average errors. Proposed_1 hardware has less area than proposed_0. It has same power consumption as proposed_0. It has higher accuracy than proposed_0. It has almost the same average error as proposed_0. But, it has larger maximum error than proposed_0. Therefore, either proposed_0 or proposed_1 hardware can be used in an application depending on its accuracy and hardware requirements.

Proposed_2 hardware is faster than proposed_0 and proposed_1 hardware. It also has less area and lower power consumption than proposed_0 and proposed_1 hardware. However, it has larger maximum and average error than proposed_0 and proposed_1 hardware. Therefore, it can be used in applications which can tolerate its maximum and average error.

Since using two 4-bit subtractors instead of one 8-bit subtractor significantly reduces the delay of critical path which is carry propogation, proposed_half hardware is the fastest approximate AD hardware. It also has less area than proposed_0, proposed_1, and proposed_2 hardware. However, it has larger maximum and average error than proposed_0, proposed_1, and proposed_2 hardware. Therefore, it can be used in applications which can tolerate its maximum and average error.

In the hardware implementations of applications using AD operations such as video compression, frame rate conversion and depth estimation, large number of parallel AD hardware such as 512, 1024 are used. In this thesis, area and power consumption results are reported for one AD hardware. Area and power consumption reductions achieved by

using the approximate AD hardware proposed in this thesis would be much larger for the hardware implementations using large number of parallel AD hardware.

5.2 Novel Approximate Constant Multiplier

Multiplying a variable with a constant is called constant multiplication. Constant multiplication is used in many applications such as video processing, video compression and machine learning. Therefore, in this thesis, a novel approximate constant multiplication technique is proposed. The proposed approximate constant multiplication technique is based on the exact constant multiplier proposed in [64] which can only be used for the DSP blocks in FPGAs. However, the proposed approximate constant multiplier can be used in both FPGA and ASIC implementations.

The proposed approximate constant multiplication technique decreases complexity of constant multiplication by converting it to a multiplication with a smaller constant, concatenation and constant shift operation. It achieves this by manipulating variable multiplicand and constant multiplier in the constant multiplication operation. Since concatenation and constant shift operations require no hardware resources, approximate constant multiplication hardware implementing the proposed approximation technique reduces constant multiplication to multiplication with a smaller constant.

Since HEVC 2D transform and VVC 2D transform algorithms include many constant multiplication operations, in this thesis, HEVC 2D transform and VVC 2D transform algorithms are selected as case studies for the proposed approximate constant multiplier. The proposed approximate constant multiplier causes negligible PSNR loss and bit rate increase when it is used to implement the constant multiplications in HEVC 2D transform and VVC 2D transform. The proposed approximate constant multiplier reduces area, reduces power consumption, and increases performance of HEVC 2D transform hardware and VVC 2D transform hardware.

5.2.1 Proposed Approximate Constant Multiplier

5.2.1.1 Proposed Approximate Constant Multiplication Technique

The proposed approximate constant multiplication technique decreases complexity of constant multiplication by converting it to a multiplication with a smaller constant, concatenation and constant shift operation. It achieves this by manipulating variable multiplicand and constant multiplier in the constant multiplication operation. Since

concatenation and constant shift operations require no hardware resources, approximate constant multiplication hardware implementing the proposed approximation technique reduces constant multiplication to multiplication with a smaller constant.

Multiplication of a v bit variable V with c bit constant C is shown in equation (5.1). Constant multiplier (C) is manipulated as in equation (5.2). Any constant integer can be written as in equation (5.2). MSBs and LSBs of variable multiplicand (V) are separated as in equation (5.3) using the b value found in equation (5.2). Then, manipulated versions of V and C are multiplied as in equations (5.4) - (5.9). Equation (5.9) implements exact constant multiplication operation. The symbols "×", "«" and "{,}" represent multiplication, left shift and concatenation operations, respectively.

$$P = V \times C \tag{5.1}$$

$$C = 2^{a} \times (1 + 2^{b} \times CC) \tag{5.2}$$

$$V = 2^{b} \times V[v-1:b] + V[b-1:0] \tag{5.3}$$

$$V \times C = V \times 2^{a} \times (1 + 2^{b} \times CC) \tag{5.4}$$

$$V \times C = 2^{a} \times (V + V \times 2^{b} \times CC) \tag{5.5}$$

$$V \times C = 2^{a} \times (2^{b} \times V[v-1:b] + V[b-1:0] + V \times 2^{b} \times CC) \tag{5.6}$$

$$V \times C = 2^{a} \times (2^{b} \times (V \times CC + V[v-1:b]) + V[b-1:0]) \tag{5.7}$$

$$V \times C = 2^{a} \times \{(V \times CC + V[v-1:b]), V[b-1:0]\} \tag{5.8}$$

$$V \times C = \{(V \times CC + V[v-1:b]), V[b-1:0]\} << a \tag{5.9}$$

The manipulated exact multiplication equation in (5.9) requires multiplication of variable multiplicand (V) with a smaller constant (CC) than the constant multiplier (C), an addition, a concatenation and a constant shift operation. Addition operation in equation (5.9) is removed to obtain the proposed approximate constant multiplication equation in (5.10).

$$V \times C = \{(V \times CC), V[b-1:0]\} << a \tag{5.10}$$

Concatenation and constant shift operations require no hardware resources. Therefore, the proposed approximation technique reduces multiplication with constant C

to multiplication with a smaller constant (CC). Computational complexity reduction depends on the values of constants C and CC. In the best case, CC is 1 and constant multiplication is eliminated. In the worst case, CC is one bit smaller than C.

Three approximate constant multiplication examples are shown in Figure 5.5. These examples show that constant CC is much smaller than constant C. Therefore, the proposed approximation technique reduces bit length of constant multiplication. It also removes addition operation. In one of the examples, since CC is 1, constant multiplication is also removed. Therefore, approximate constant multiplication hardware implementing the proposed approximation technique performs multiplication with constant 36 without using any hardware resources.

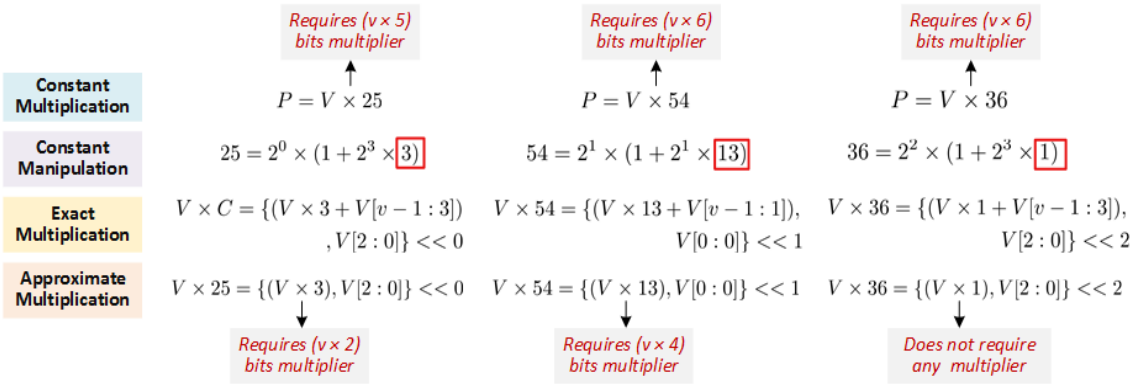

Figure 5.5 Examples of Approximate Constant Multiplication

Exact constant multiplication hardware, exact constant multiplication hardware with proposed manipulation and the proposed approximate constant multiplication hardware are shown in Figure 5.6. The symbols “v”, “c” and “cc” represent bit lengths of input variable (V), constant (C) and manipulated constant (CC), respectively. Since CC is always smaller than C, the proposed approximation technique reduces area and increases performance of constant multiplication hardware.

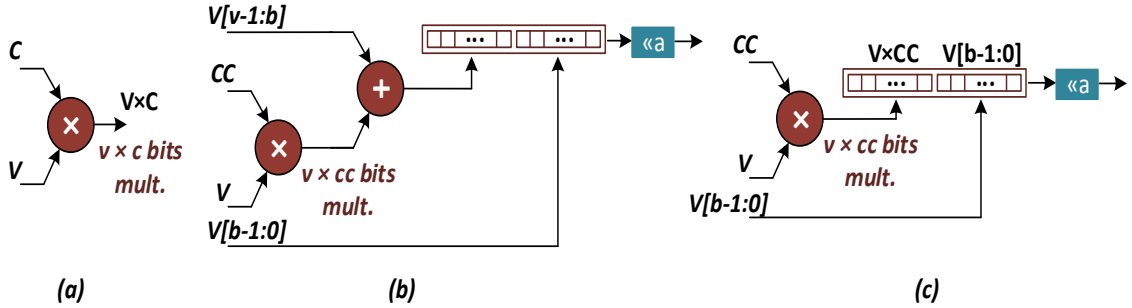

Figure 5.6 Constant Multiplication Hardware (a) Exact Constant Multiplication, (b) Exact Constant Multiplication with Proposed Manipulation, (c) Proposed Approximate Constant Multiplication

5.2.1.2 Proposed Approximate Constant Multiplier Datapath Generator

The proposed approximate constant multiplication requires pre-determined constant multiplication, concatenation and constant shift operations. These operations differ for each constant C. They should be determined for implementing the datapath necessary to perform the approximate constant multiplication.

As shown in Figure 5.7, a python based datapath generator is proposed to determine constant multiplication, concatenation and constant shift operations for an input variable and constants. The proposed datapath generator takes input variable (V) bit length and constants that will be multiplied with V as inputs. If a constant is power of 2, this constant multiplication is implemented with a constant shift operation. If a constant is power of 2 multiple of another constant in the input constants, this constant multiplication is also implemented with constant shift operation.

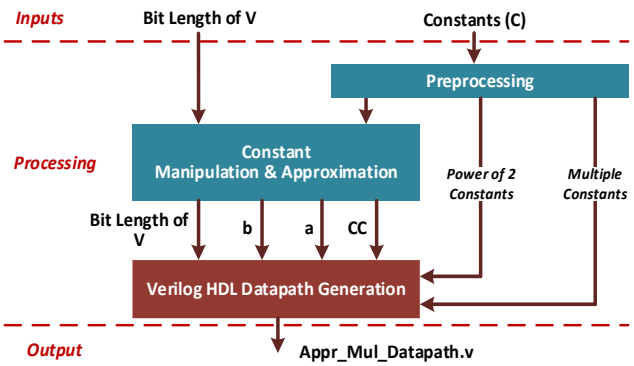

Figure 5.7 Flow Chart of the Proposed Datapath Generator

The remaining constant multiplications are implemented using the proposed approximate constant multiplication technique. The proposed datapath generator determines constant multiplication, concatenation and constant shift operations necessary for these input constants. Then, it generates a text file containing Verilog HDL implementations of the datapaths which perform the constant multiplications.

5.2.2 Case Studies: HEVC 2D Transform and VVC 2D Transform

High Efficiency Video Coding (HEVC) and Versatile Video Coding (VVC) video compression standards use Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST) for 2D transform operations [65]-[66]. Since DCT and DST algorithms include many constant multiplication operations, HEVC 2D transform and VVC 2D transform algorithms are selected as case studies for the proposed approximate constant multiplication technique.

HEVC uses DCT-II for transform operations. It uses 4x4, 8x8, 16x16, 32x32 Transform Unit (TU) sizes. HEVC uses DST-VII only for 4x4 TUs in certain cases. HEVC performs 2D transform operation by applying 1D transforms in vertical and horizontal directions. The coefficients in HEVC 1D transform matrices are derived from DCT and DST basis functions. However, integer coefficients are used for simplicity. 4x4 DCT matrix used in HEVC is shown in equation (5.11) as an example.

$$DCT\_4x4_{HEVC} = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix} \tag{5.11}$$

VVC uses DCT-II, DCT-VIII and DST-VII for transform operations. It uses 4x4, 8x8, 16x16, 32x32 and 64x64 TU sizes. VVC performs 2D transform operation by applying 1D transforms in vertical and horizontal directions. While HEVC uses the same transform types in vertical and horizontal directions, VVC may use different transform types in vertical and horizontal directions. The coefficients in VVC 1D transform matrices are derived from DCT and DST basis functions. However, integer coefficients are used for simplicity. 4x4 DCT-V matrix used in VVC is shown in equation (5.12) as an example.

$$DCT\_4x4_{VVC} = \begin{bmatrix} 117 & 219 & 296 & 336 \\ 296 & 296 & 0 & -296 \\ 336 & -117 & -296 & 219 \\ 219 & -336 & 296 & -117 \end{bmatrix} \tag{5.12}$$

29 different constants (C values) used in HEVC DCT matrices are listed in Table 5.3. CC, a and b values determined to manipulate these constants as in equation (5.2) and the corresponding approximate constant multiplication equations as in equation (5.10) are also listed in Table 5.3. Multiplications with constants 4 and 64 are implemented exactly by constant shift operations. Multiplication with same constant in the approximate constant multiplication equations is implemented once and the result is used for all equations. For example, multiplication with 5 is implemented once and the result is used for multiplications with constants 22, 82 and 88.

Table 5.3 Approximate Constant Multiplications for HEVC 2D DCT

C	CC	a	b	Approximate constant multiplication
4	-	-	-	V<<2
9	1	0	3	{V, V[2:0]} << 0
13	3	0	2	{(V * 3), V[1:0]} << 0
18	1	1	3	{V, V[2:0]} << 1
22	5	1	1	{(V * 5), V[0:0]} << 1
25	3	0	3	{(V * 3), V[2:0]} << 0
31	15	0	1	{(V * 15), V[0:0]} << 0
36	1	2	3	{V, V[2:0]} << 2
38	9	1	1	{(V * 9), V[0:0]} << 1
43	21	0	1	{(V * 21), V[0:0]} << 0
46	11	1	1	{(V * 11), V[0:0]} << 1
50	3	1	3	{(V * 3), V[2:0]} << 1
54	13	1	1	{(V * 13), V[0:0]} << 1
57	7	0	3	{(V * 7), V[2:0]} << 0
61	15	0	2	{(V * 15), V[1:0]} << 0
64	-	-	-	V<<6
67	33	0	1	{(V * 33), V[0:0]} << 0
70	17	1	1	{(V * 17), V[0:0]} << 1
73	9	0	3	{(V * 9), V[2:0]} << 0
75	37	0	1	{(V * 37), V[0:0]} << 0
78	19	1	1	{(V * 19), V[0:0]} << 1
80	1	4	2	{V, V[1:0]} << 4
82	5	1	3	{(V * 5), V[2:0]} << 1
83	41	0	1	{(V * 41), V[0:0]} << 0
85	21	0	2	{(V * 21), V[1:0]} << 0
87	43	0	1	{(V * 43), V[0:0]} << 0
88	5	3	1	{(V * 5), V[0:0]} << 3
89	11	0	3	{(V * 11), V[2:0]} << 0
90	11	1	2	{(V * 11), V[1:0]} << 1

Approximate constant multiplications for 57 different constants used in VVC transforms are also implemented. However, they are not shown in this thesis for simplicity.

5.2.2.1 Error Analysis

Error caused by proposed approximate constant multiplier differs for each constant. Errors caused for the constants used in HEVC 2D DCT are determined as follows. Average percentage error for a constant C is calculated as in equations (5.13)-(5.15). Input variable bit length is taken as 8 bits. The constant is multiplied with all possible values of input variable (0-255) with exact multiplier and with the proposed approximate constant multiplier. Error for the input variable value i (E_i) is calculated by taking absolute difference of the exact multiplication result and approximate multiplication result as in equation (5.13). Percentage error for the input variable value i (PE_i) is calculated as in equation (5.14). Average percentage error for the constant C is calculated by computing average of percentage errors for all possible values of input variable (0-255) as in equation (5.15).

$$E_i = \left| exact\ (C \times i) - appr\ (C \times i) \right| \tag{5.13}$$

$$PE_i = \frac{E_i}{exact\ (C \times i)} \times 100 \tag{5.14}$$

$$average\ \ percentage\ \ error = \frac{\sum_{i=0}^{255} PE_i}{256} \tag{5.15}$$

Average percentage errors for the constants used in HEVC 2D DCT are calculated and shown in Figure 5.8. The results show that the proposed approximate constant multiplier causes very small errors. Average percentage errors for the constants used in VVC 2D transform are also calculated. However, they are not shown in this thesis for simplicity. The proposed approximate constant multiplier causes very small errors for these constants as well.

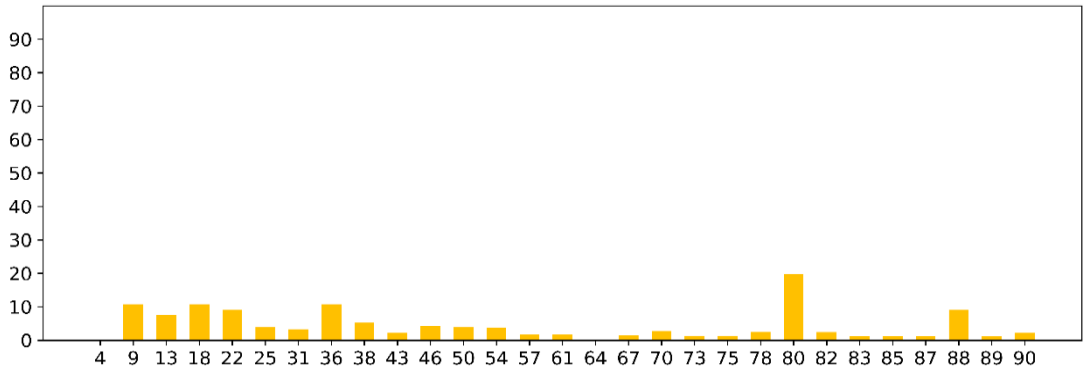

Figure 5.8 Average Percentage Error (%) for HEVC 2D DCT Constants

Impacts of the proposed approximate constant multiplier and the approximate multipliers proposed in [61], [62] and [63] on rate-distortion performance of HEVC standard is determined using HEVC HM reference software encoder 15.0 [34]. First frame of Basketball Drive (1920x1080), Kristen and Sara (1280x720), and Party Scene (832x480) test videos are coded with HEVC HM 15.0 using five different multipliers for implementing constant multiplications in HEVC 2D transform; exact multiplier (Orig_H), the approximate constant multiplier proposed in this thesis (Prop_H), the approximate multiplier proposed in [61] (M1_H), the approximate multiplier proposed in [62] (M2_H), and the approximate multiplier proposed in [63] (M3_H).

The resulting rate-distortion performances are shown in Figure 5.9. The proposed approximate constant multiplier causes negligible PSNR loss and bit rate increase compared to using exact multiplier. The proposed approximate constant multiplier has better rate-distortion performance than the approximate multipliers proposed in the literature.

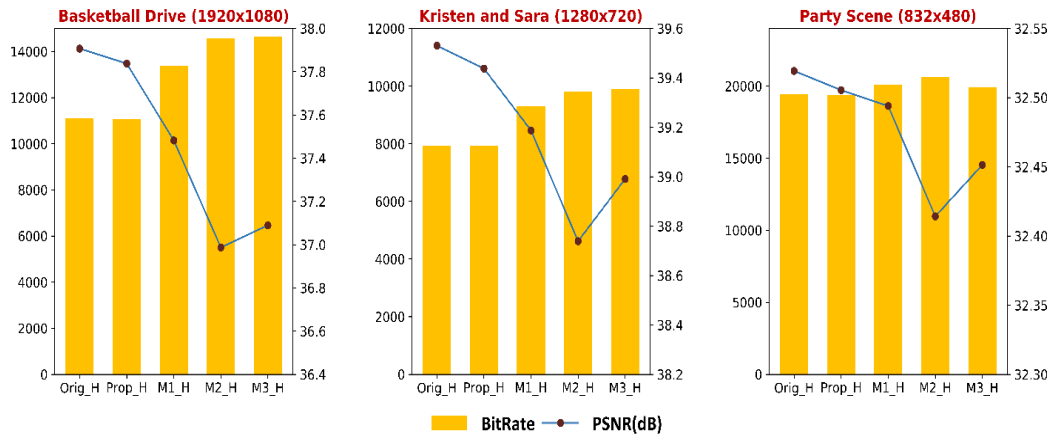

Figure 5.9 HEVC Bit Rate and PSNR (dB) Comparison

Impacts of the proposed approximate constant multiplier and the approximate multipliers proposed in [61], [62] and [63] on rate-distortion performance of VVC standard is determined using VVC VTM reference software encoder 2.0 [67]. First frame of Basketball Drive (1920x1080), Kristen and Sara (1280x720), and Party Scene (832x480) test videos are coded with VVC VTM 2.0 using five different multipliers for implementing constant multiplications in VVC 2D transform; exact multiplier (Orig_V), the approximate constant multiplier proposed in this thesis (Prop_V), the approximate multiplier proposed in [61] (M1_V), the approximate multiplier proposed in [62] (M2_V), and the approximate multiplier proposed in [63] (M3_V).

The resulting rate-distortion performances are shown in Figure 5.10. The proposed approximate constant multiplier causes negligible PSNR loss and bit rate increase compared to using exact multiplier. The proposed approximate constant multiplier has better rate-distortion performance than the approximate multipliers proposed in the literature.

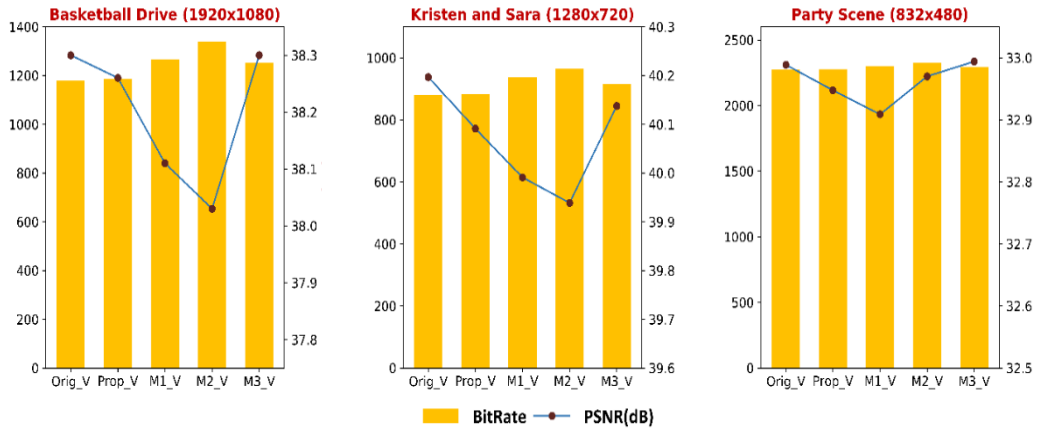

Figure 5.10 VVC Bit Rate and PSNR (dB) Comparison

5.2.2.2 Proposed Hardware Implementations

Five different HEVC 2D transform hardware are designed and implemented. The only difference between them is the multipliers used to implement constant multiplications in HEVC 2D transform. First hardware (Orig_H) uses exact multiplier. Second hardware (Prop_H) uses the approximate constant multiplier proposed in this thesis. The other three hardware (M1_H, M2_H, M3_H) use the approximate multipliers proposed in [61], [62], [63], respectively.

Five different VVC 2D transform hardware are also designed and implemented. The only difference between them is the multipliers used to implement constant multiplications in VVC 2D transform. First hardware (Orig_V) uses exact multiplier. Second hardware (Prop_V) uses the approximate constant multiplier proposed in this thesis. The other three hardware (M1_V, M2_V, M3_V) use the approximate multipliers proposed in [61], [62], [63], respectively.

The proposed HEVC and VVC 2D transform hardware perform 2D transform by first performing 1D transform on the columns of a TU, and then performing 1D transform on the rows of the TU. After 1D column transform, the resulting coefficients are stored in a transpose memory, and they are used as input for 1D row transform. The proposed HEVC 2D transform hardware support 4x4, 8x8, 16x16 and 32x32 TUs. The proposed VVC 2D transform hardware support 4x4, 8x8, 16x16, 32x32 and 64x64 TUs.

The proposed five HEVC 2D transform hardware and five VVC 2D transform hardware are implemented using Verilog HDL. The Verilog RTL codes are synthesized and mapped to a Xilinx XC7VX690T FFG1761 FPGA with speed grade 3 using Xilinx Vivado 2017.2. FPGA implementations are verified with post place and route simulations. Post place and route simulation results matched the results of HEVC 2D transform and VVC 2D transform software implementations. The FPGA implementation results are shown in Table 5.4 and Table 5.5.

Table 5.4 FPGA Implementation Results of HEVC 2D Transform

	Orig_H	Prop_H	M1_H	M2_H	M3_H
FPGA	Virtex-7	Virtex-7	Virtex-7	Virtex-7	Virtex-7
LUT	31062	30986	42553	47266	45571
DFF	11862	11648	11543	12174	11893
BRAM	32	32	32	32	32
DSP Block	370	108	0	0	20
Frequency (MHz)	147	161	149	147	158

Table 5.5 FPGA Implementation Results of VVC 2D Transform

	Orig_V	Prop_V	M1_V	M2_V	M3_V
FPGA	Virtex-7	Virtex-7	Virtex-7	Virtex-7	Virtex-7
LUT	100279	83424	133641	141649	145544
DFF	32336	25444	46190	51062	47535
BRAM	32	32	32	32	32
DSP Block	1303	240	0	0	20
Frequency (MHz)	117	109	109	104	108

As shown in Table 5.4, HEVC 2D transform FPGA implementation using the proposed approximate constant multiplier (Prop_H) has 70.8% less DSP blocks than HEVC 2D transform FPGA implementation using exact multiplier (Orig_H). Prop_H FPGA implementation also has higher performance than Orig_H FPGA implementation. Prop_H FPGA implementation has less Lookup Tables (LUT) and higher performance than HEVC 2D transform FPGA implementations using the approximate multipliers proposed in the literature (M1_H, M2_H, M3_H). However, Prop_H FPGA implementation has DSP blocks. M1_H, M2_H, M3_H FPGA implementations do not have DSP blocks.

As shown in Table 5.5, VVC 2D transform FPGA implementation using the proposed approximate constant multiplier (Prop_V) has 81.5% less DSP blocks than VVC 2D transform FPGA implementation using exact multiplier (Orig_V). Prop_V FPGA implementation has less LUTs than VVC 2D transform FPGA implementations using the approximate multipliers proposed in the literature (M1_V, M2_V, M3_V). However, Prop_V FPGA implementation has DSP blocks. M1_V, M2_V, M3_V FPGA implementations do not have DSP blocks.

Power consumptions of all HEVC and VVC 2D transform FPGA implementations are estimated using Xilinx Vivado 2017.2. Signal activities captured during post place and route timing simulations are used to estimate power consumptions. Energy consumptions of HEVC 2D transform FPGA implementations for transforming six 4x4 TUs, four 8x8 TUs, four 16x16 TUs, five 32x32 TUs are determined and shown in Figure 5.11. Prop_H FPGA implementation has less energy consumption than the other HEVC 2D transform FPGA implementations. Energy consumptions of VVC 2D transform FPGA implementations for transforming two 4x4 TUs, two 8x8 TUs, two 16x16 TUs, three 32x32 TUs are determined and shown in Figure 5.12. Prop_V FPGA implementation has less energy consumption than the other VVC 2D transform FPGA implementations.

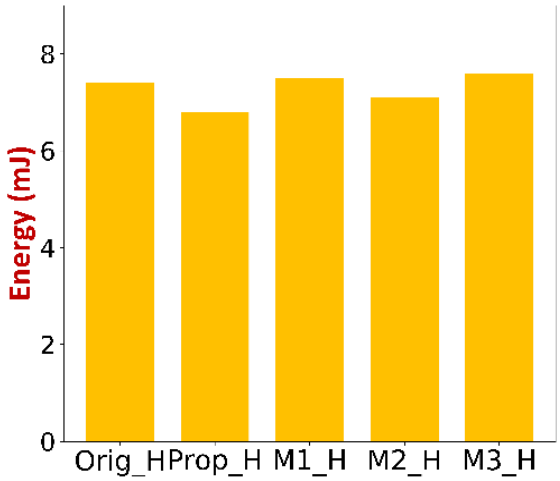

Figure 5.11 Energy Consumptions of HEVC 2D Transform FPGA Implementations

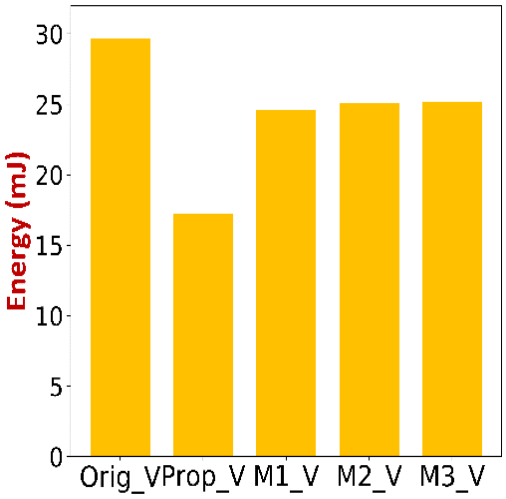

Figure 5.12 Energy Consumptions of VVC 2D Transform FPGA Implementations

The Verilog RTL codes are also synthesized, placed and routed to a TSMC 90nm standard cell library. The ASIC implementation results are shown in Table 5.6 and Table 5.7. Gate counts of all the ASIC implementations are calculated according to NAND (3x1) gate area. As shown in Table 5.6, HEVC 2D transform ASIC implementation using the proposed approximate constant multiplier (Prop_H) has smaller area, lower power consumption and higher performance than the other HEVC 2D transform ASIC implementations. As shown in Table 5.7, VVC 2D transform ASIC implementation using the proposed approximate constant multiplier (Prop_V) has smaller area and lower power consumption than the other VVC 2D transform ASIC implementations.

Table 5.6 ASIC Implementation Results of HEVC 2D Transform

	Orig_H	Prop_H	M1_H	M2_H	M3_H
Technology	TSMC 90 nm	TSMC 90 nm	TSMC 90 nm	TSMC 90 nm	TSMC 90 nm
Area	180 K	152 K	180 K	195 K	187 K
Frequency (MHz)	250	330	278	264	264
Power (mW)	102	90.6	105.4	115.2	112

Table 5.7 ASIC Implementation Results of VVC 2D Transform

	Orig_V	Prop_V	M1_V	M2_V	M3_V
Technology	TSMC 90 nm	TSMC 90 nm	TSMC 90 nm	TSMC 90 nm	TSMC 90 nm
Area	630 K	458 K	632 K	672 K	646 K
Frequency (MHz)	192	250	250	245	260
Power (mW)	332.2	248	333.1	358.8	356.7

CHAPTER VI

CONCLUSIONS AND FUTURE WORKS

In this thesis, we proposed a novel computation and energy reduction technique for HEVC intra prediction. We designed low energy, reconfigurable HEVC intra prediction hardware using the proposed technique. We also designed an FPGA implementation of HEVC intra prediction using DSP blocks. We proposed reconfigurable VVC intra prediction hardware. We also designed an FPGA implementation of VVC intra prediction using DSP blocks. We proposed VVC fractional interpolation hardware. We proposed several approximate absolute difference hardware. We proposed a novel approximate constant multiplier. We designed HEVC 2D transform and VVC 2D transform hardware using the proposed approximate constant multiplier.

We quantified computation reductions achieved by the proposed techniques and video quality loss caused by the proposed approximation techniques. The proposed approximation techniques cause very small PSNR loss. The other proposed techniques cause no PSNR loss. We implemented the proposed hardware architectures in Verilog HDL. We mapped the Verilog RTL codes to Xilinx Virtex 6 or Xilinx Virtex 7 FPGAs, and we estimated their power consumptions using Xilinx XPower Analyzer tool. The proposed techniques significantly reduced power and energy consumptions of these FPGA implementations.

As future work, approximate video compression algorithms for HEVC and VVC video compression standards can be proposed. HEVC and VVC video encoders and decoders can be proposed by implementing exact or approximate hardware of HEVC and VVC video compression algorithms and integrating them with the ones implemented in this thesis. VVC video compression standard is still in standardization process. The

proposed techniques can be used to implement the video compression algorithms in final VVC standard.

BIBLIOGRAPHY

[1] High Efficiency Video Coding, ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), ITU-T and ISO/IEC, April 2013.

[2] G.J. Sullivan, J.R. Ohm, W.J. Han, T.Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[3] F. Pescador, M. Chavarrias, M.J. Garrido, E. Juarez, C. Sanz, "Complexity Analysis of an HEVC Decoder Based on a Digital Signal Processor," *IEEE Trans. On Consumer Electronics*, vol. 59, no. 2, pp. 391-399, May 2013.

[4] Algorithm Description of Versatile Video Coding and Test Model 5, Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG11, March 2019.

[5] Q. Xu, T. Mytkowicz and N. S. Kim, "Approximate Computing: A Survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8-22, Feb. 2016.

[6] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, J. Henkel, "Cross-Layer approximate computing: From logic to architectures," *IEEE Design Automation Conference*, Austin, USA, Jun. 2016.

[7] C. Liu, J. Han, F. Lombardi, "An analytical framework for evaluating the error characteristics of approximate adders," *IEEE Trans. on Computers*, vol. 64, no. 5, pp. 1268-1281, May 2015.

[8] E. Kalali and I. Hamzaoglu, "Approximate HEVC fractional interpolation filters and their hardware implementations," *IEEE Trans. on Consumer Electronics*, vol. 64, no. 3, pp. 285-291, Aug. 2018.

[9] O. Tasdizen, A. Akin, H. Kukner, and I. Hamzaoglu, "Dynamically variable step search motion estimation algorithm and a dynamically reconfigurable hardware for its implementation", *IEEE Trans. on Consumer Electronics*, vol. 55, no. 3, pp.1645-1653, Aug. 2009.

[10] M. Cetin and I. Hamzaoglu, "An adaptive true motion estimation algorithm for frame rate conversion of high definition video and its hardware implementations," *IEEE Trans. on Consumer Electronics*, vol. 57, no. 2, pp. 923-931, May 2011.

[11] S. Jin *et al.*, "FPGA Design and Implementation of a Real-Time Stereo Vision System," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15-26, Jan. 2010.

[12] J. Lainema, F. Bossen, W. J. Han, J. Min and K. ugur, "Intra Coding of the HEVC Standard", *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1792-1801, Dec. 2012.

[13] J. Chen, E. Alshina, G. J. Sullivan, J. R. Ohm, J.Boyce, "Algorithm Description of Joint Exploration Model 4", *JVET-D1001*, Oct. 2016.

[14] J. Chen, Y.Chen, M. Karczewicz, X. Li, W. Chien, "Enhanced Multiple Transform for Video Coding", *Proc. Data Compression Conference*, April 2016.

[15] X. Zhao, J. Chen, M. Karczewics, L. Zhang, X. Li, W.Chien, "Enhanced Multiple Transform for Video Coding", *Proc. Data Compression Conference*, April 2016.

[16] T. Biatek, V. Lorcy, P. Castel, P. Philippe, "Low-Complexity Adaptive Multiple Transform for Video Coding", *Proc. Data Compression Conference*, April 2016.

[17] J. Vanne, M. Viitanen, T. D. H*ämäläinen and A. Hallapuro, "Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs", IEEE Trans. On Circuits and Systems for Video Technology, vol.22, no. 12, pp. 1885-1898, Dec. 2012.*

[18] H. Azgin, E. Kalali, I. Hamzaoglu, "A Computation and Energy Reduction Technique for HEVC Intra Prediction", *IEEE Trans. On Consumer Electronics*, vol. 63, no. 1, pp. 36-43, Feb. 2017.

[19] H. Azgin, A. C. Mert, E. Kalali and I. Hamzaoglu, "An Efficient FPGA Implementation of HEVC Intra Prediction", *IEEE Int. Conf. on Consumer Electronics*, March 2018.

[20] H. Azgin, A. C. Mert, E. Kalali and I. Hamzaoglu, "Reconfigurable Intra Prediction Hardware for Future Video Coding", *IEEE Trans. on Consumer Electronics*, vol. 63, no. 4, pp. 419-425, Nov. 2017.

[21] H. Azgin, E. Kalali and I. Hamzaoglu, "An Efficient FPGA Implementation of Versatile Video Coding Intra Prediction", *Euromicro Conf. on Digital System Design (DSD)*, Aug. 2019.

[22] H. Azgin, A. C. Mert, E. Kalali and I. Hamzaoglu, “A Reconfigurable Fractional Interpolation Hardware for VVC Motion Compensation” *Euromicro Conf. on Digital System Design (DSD),* Aug. 2018.

[23] A. C. Mert, H. Azgin, E. Kalali and I. Hamzaoglu, “Novel Approximate Absolute Difference Hardware”, *Euromicro Conf. on Digital System Design (DSD),* Aug. 2019.

[24] B. Min, Z. Xu, R. C. C. Cheung, “A Fully Pipelined Hardware Architecture for Intra Prediction of HEVC”, *IEEE Trans. on Circuits and Systems for Video Technology,* July 2016.

[25] M. Abeydeera, M. Karunaratne, G. Karunaratne, K. De Silva, A. Pasqual, “4K Real Time HEVC Decoder on FPGA”, *IEEE Trans. on Circuits and Systems for Video Technology,* vol. 26, no. 1, pp. 236-249, Jan. 2016.

[26] F. Amish, E. B. Bourennane, “Fully Pipelined Real Time Hardware Solution for High Efficiency Video Coding (HEVC) Intra Prediction”, *Journal of System Architecture,* vol. 64, pp. 133-147, March 2016.

[27] M. U. K. Khan, M. Shafique, M. Grellert, J. Henkel, “Hardware-Software Collaborative Complexity Reduction Scheme for The Emerging HEVC Intra Encoder,” *Design, Automation and Test in Europe (DATE) Conference,* pp. 125-128, March 2013.

[28] G. Pastuszak, A. Abramowski, “Algorithm and Architecture Design of The H.265/HEVC Intra Encoder”, *IEEE Trans. on Circuits and Systems for Video Technology,* vol. 26, no. 1, pp. 210-222, Jan. 2016.

[29] C. T. Huang, M. Tikekar, A. Chandrakasan, “Memory-Hierarchical and Mode-Adaptive HEVC Intra Prediction Architecture for Quad Full HD Video Decoding”, *IEEE Trans. on Very Large Scale Integration (VLSI) Systems,* vol. 22, no. 7, pp. 1515-1525, July 2014.

[30] P. Chiang, Y. Ting, H. Chen, S. Jou, I. Chen, H. Fang, T. Chang, “A QFHD 30 fps HEVC Decoder Design”, *IEEE Trans. on Circuits and Systems for Video Technology,* vol. 26, no. 4, pp. 724-735, April 2016.

[31] N. Zhou, D. Ding, L. Yu, “On Hardware Architecture and Processing Order of HEVC Intra Prediction Module”, *Picture Coding Symposium,* pp. 101-104, Dec. 2013.

[32] Z. Liu, D. Wang, H. Zhu, X. Huang, “41.7BN-pixels/s Reconfigurable Intra Prediction Architecture for HEVC 2560x1600 Encoder”, *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 2634-2638, May 2013.

[33] E. Kalali, Y. Adibelli, I. Hamzaoglu, “A Low Energy Intra Prediction Hardware for High Efficiency Video Coding”, *Journal of Real-Time Image Processing*, Dec. 2014.

[34] K. McCann, B. Bross, W.J. Han, I.K. Kim, K. Sugimoto, G. J. Sullivan, “High Efficiency Video Coding (HEVC) Test Model 15 (HM 15) Encoder Description”, *JCTVC-Q1002*, June 2014.

[35] F. Bossen, “Common test conditions and software reference configurations”, *JCTVC-I1100*, May 2012.

[36] E. Kalali, Y. Adibelli, I. Hamzaoglu, “A High Performance and Low Energy Intra Prediction Hardware for High Efficiency Video Coding”, *Int. Conf. on Field Programmable Logic and Applications (FPL),* pp. 719-722, Aug. 2012.

[37] J. Chen, E. Alshina, G. J. Sullivan, J. R. Ohm, J. Boyce, “Algorithm Description of Joint Exploration Model 7,” *JVET-G1001,* Jul. 2017.

[38] E. Kalali, Y. Adibelli, and I. Hamzaoglu, “A reconfigurable HEVC sub-pixel interpolation hardware”, *IEEE Int. Conference on Consumer Electronics - Berlin,* Sept. 2013.

[39] E. Kalali and I. Hamzaoglu, “A low energy HEVC sub-pixel interpolation hardware,” *IEEE Int. Conference on Image Processing,* pp. 1218-1222, Oct. 2014.

[40] G. Pastuszak and M. Trochimiuk, “Architecture design and efficiency evaluation for the high-throughput interpolation in the HEVC encoder”, *16th Euromicro Conference on Digital System Design,* Sep. 2013.

[41] C. M. Diniz, M. Shafique, S. Bampi, and J. Henkel, “A reconfigurable hardware architecture for fractional pixel interpolation in high efficiency video coding,” IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 2, pp. 238-251, Feb. 2015.

[42] H. Maich, C. Afonso, D. Franco, B. Zatt, M. Porto, and L. Agostini, “High throughput hardware design for the HEVC fractional motion estmation interpolation unit”, IEEE 20th International Conference on Electronics, Circuits, and Systems, May 2014.

[43] A. C. Mert, E. Kalali, and I. Hamzaoglu, “An HEVC fractional interpolation hardware using memory based constant multiplication,” *IEEE Int. Conf. On Consumer Electronics (ICCE),* pp. 742-746, Jan. 2018.

[44] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” 18th IEEE European Test Symposium (ETS), May 2013.

[45] S. Mittal, “A survey of techniques for approximate computing,” ACM Computing Surveys, vol. 48, no. 4, May 2016.

[46] G. A. Gillani, M. A. Hanif, M. Krone, S. H. Gerez, M. Shafique, and A. B. J. Kokkeler, “SquASH: Approximate square-accumulate with self-healing,” IEEE Access, vol. 6, pp. 49112-49128, Aug. 2018.

[47] T. Ayhan and M. Altun, “Circuit aware approximate system design with case studies in image processing and neural networks,” IEEE Access, vol. 7, pp. 4726-4734, Dec. 2018.

[48] I. Hammad and K. El-Sankary, “Impact of approximate multipliers on VGG deep learning network,” IEEE Access, vol. 6, pp. 60438-60444, Oct. 2018.

[49] B. Liu, H. Qin, Y. Gong, W. Ge, M. Xia, and L Shi, “EERA-ASR: An energy-efficient reconfigurable architecture for automatic speech recognition with hybrid DNN and approximate computing,” IEEE Access, vol. 6, pp. 52227-52237, Sep. 2018.

[50] K. Roy and A. Raghunathan, “Approximate computing: An energy-efficient computing technique for error resilient applications,” IEEE Computer Society Annual Symposium on VLSI, Jul. 2015.

[51] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-power digital signal processing using approximate adders,” IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 1, pp. 124-137, Jan. 2013.

[52] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. L. Kamran, “AxBench: A multiplatform benchmark suite for approximate computing,” IEEE Design & Test, vol. 34, no. 2, pp. 60-68, Apr. 2017.

[53] E. Kalali and I. Hamzaoglu, “Approximate HEVC fractional interpolation filters and their hardware implementations,” IEEE Trans. on Consumer Electronics, vol. 64, no. 3, pp. 285-291, Aug. 2018.

[54] S. Xu and B. C. Schafer, “Toward self-tunable approximate computing,” IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 4, pp. 778-789, Apr. 2019.

[55] Y. Kim, Y. Zhang, and P. Li, “Energy efficient approximate arithmetic for error resilient neuromorphic computing,” IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 11, pp. 2733-2737, Nov. 2015.

[56] A. Raha, H. Javakumar, and V. Raghunathan, "Input-based dynamic reconfiguration of approximate arithmetic units for video encoding," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 3, pp. 846-857, Mar. 2016.

[57] V. T. Lee, A. Alaghi, R. Pamula, V. S. Sathe, L. Ceze, and M. Oskin, "Architecture consideration for stochastic computing accelerators," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2277-2289, Nov. 2018.

[58] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," Design, Automation & Test in Europe Conference & Exhibition (DATE), Munich, Germany, April 2008.

[59] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," 12th Int. Symposium Integrated Circuit, Singapore, Singapore, Dec. 2009.

[60] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, USA, Jun. 2015.

[61] A. Momeni, J. Han, P. Montuschi, F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Computers,* Vol. 64, No. 4, pp. 984-994, Apr. 2015.

[62] K. Bhardwaj, P. S. Mane, J. Henkel, "Power- and area-efficient Approximate Wallace Tree Multiplier for error-resillient systems," in *Proc. 15th Int. Symp. Quality Electron. Design (ISQED),* Mar. 2014, pp. 263-269.

[63] T. Yang, T. Ukezono, and T. Sato, "A low-power high-speed accuracy-controllabel approximate multiplier design," *23rd Asia and South Pacific Design Automation Conf. (ASP-DAC),* Jan. 2018.

[64] A. C. Mert, H. Azgin, E. Kalali, and I. Hamzaoglu, "Efficient multiple constant multiplication using DSP blocks in FPGA," *28th Int. Conf. on Field Programmable Logic and Applications (FPL),* Aug. 2018.

[65] E. Kalali, A. C. Mert, and I. Hamzaoglu, "A computation and energy reduction technique for HEVC Discrete Cosine Transform," *IEEE Trans. on Consumer Electronics,* vol. 62, no. 2, pp. 166-174, May 2016.

[66] A. C. Mert, E. Kalali, and I. Hamzaoglu, "High performance 2D transform hardware for future video coding," *IEEE Trans. on Consumer Electronics,* vol. 63, no. 2, pp. 117-125, May 2017.

[67] J. Chen, Y. Ye, and S. H. Kim, “Algorithm description for versatile video coding and test model 2 (VTM 2),” JVET-K1002, Jul. 2018.