

HIGH PERFORMANCE HEVC AND FVC VIDEO COMPRESSION HARDWARE
DESIGNS

by
Ahmet Can Mert

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of
the requirements for the degree of
Master of Sciences

Sabancı University

August 2017

HIGH PERFORMANCE HEVC AND FVC VIDEO COMPRESSION
HARDWARE DESIGNS

APPROVED BY:

Assoc. Prof. Dr. İlker Hamzaoğlu
(Thesis Supervisor)


.....

Assist. Prof. Dr. Kamer Kaya


.....

Assist. Prof. Dr. Erdinç Öztürk


.....

DATE OF APPROVAL: 01/08/2017

© Ahmet Can Mert 2017
All Rights Reserved

To my Family and my Love

ACKNOWLEDGEMENT

First of all, I would like to thank my supervisor, Dr. İlker Hamzaoğlu for all his guidance, support, and patience throughout my studies. I appreciate very much for his suggestions, detailed reviews and invaluable advices. It has been a great honor for me to work under his guidance. I feel myself privileged as his student.

I want to thank all members of “System-on-Chip Design and Test Lab”; Ercan Kalalı, Hasan Azgın and Firas Abdul Ghani for their great friendship and their collaboration during my studies. I want to especially thank Ercan Kalalı for sharing his experiences and his support for my studies.

I want to thank my friends; Ozan Özdenizci, Abdurrahman Burak, Onur Biricik and many others for their valuable friendship.

Deepest gratitude to my family and my love Gülizar. This thesis is dedicated with love to them for their constant support and encouragement for going through my tough periods with me.

Finally, I would like to acknowledge Sabanci University and Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting me with scholarships throughout my studies. This thesis was supported by TÜBİTAK under the contract 115E290.

HIGH PERFORMANCE HEVC AND FVC VIDEO COMPRESSION HARDWARE DESIGNS

Ahmet Can Mert

Electronics, MS Thesis, 2017

Thesis Supervisor: Assoc. Prof. İlker HAMZAOĞLU

Keywords: HEVC, Sub-Pixel Motion Estimation, Fractional Interpolation, FVC, 2D Transform

ABSTRACT

High Efficiency Video Coding (HEVC) is the current state-of-the-art video compression standard developed by Joint collaborative team on video coding (JCT-VC). HEVC has 50% better compression efficiency than H.264 which is the previous video compression standard. HEVC achieves this video compression efficiency by significantly increasing the computational complexity. Therefore, in this thesis, we proposed a low complexity HEVC sub-pixel motion estimation (SPME) technique for SPME in HEVC encoder. We designed and implemented a high performance HEVC SPME hardware implementing the proposed technique. We also designed and implemented an HEVC fractional interpolation hardware using memory based constant multiplication technique for both HEVC encoder and decoder.

Future Video Coding (FVC) is a new international video compression standard which is currently being developed by JCT-VC. FVC offers much better compression efficiency than the state-of-the-art HEVC video compression standard at the expense of much higher computational complexity. In this thesis, we designed and implemented three different high performance FVC 2D transform hardware. The proposed hardware is verified to work correctly on an FPGA board.

YÜKSEK PERFORMANSLI HEVC VE FVC VIDEO SIKIŞTIRMA DONANIM TASARIMLARI

Ahmet Can Mert

Elektronik Müh., Yüksek Lisans Tezi, 2017

Tez Danışmanı: Doç. Dr. İlker HAMZAOĞLU

Anahtar Kelimeler: HEVC, Ara Piksel Hassaslığında Hareket Tahmini, Ara Piksel Hesaplama, FVC, 2B Dönüşüm

ÖZET

Yüksek verimli video kodlama (HEVC) Joint Collaborative Team on Video Coding (JCT-VC) tarafından geliştirilen günümüzde kullanılan video sıkıştırma standardıdır. HEVC bir önceki H.264 standardına göre 50% daha iyi performans sağlamaktadır. HEVC bu video sıkıştırma verimini hesaplama karmaşıklığını önemli ölçüde artırarak başarmaktadır. Bu nedenle, bu tezde HEVC video kodlayıcısı için kullanılan ara piksel hassaslığında hareket tahmini (SPME) için düşük karmaşıklıkta HEVC SPME tekniği önerildi. Önerilen tekniği uygulayan yüksek performanslı HEVC SPME donanımı tasarlandı ve gerçekleştirildi. Ayrıca, HEVC video kodlayıcı ve kod çözücü için bellek bazlı sabit çarpma tekniği kullanan HEVC ara pikselleri oluşturma donanımı tasarlandı ve gerçekleştirildi.

Gelecek video kodlama (FVC) JCT-VC tarafından halihazırda geliştirilen yeni bir video sıkıştırma standardıdır. FVC daha fazla hesaplama karmaşıklığı pahasına günümüzde kullanılan HEVC video sıkıştırma standardından daha iyi sıkıştırma verimliliği sunmaktadır. Bu tezde, üç farklı yüksek performanslı FVC 2B dönüşüm donanımı tasarlandı ve gerçekleştirildi. Önerilen donanımın gerektiği şekilde çalıştığı FPGA'de doğrulandı.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	V
1 ABSTRACT.....	VI
2 ÖZET	VII
3 TABLE OF CONTENTS.....	VIII
LIST OF FIGURES	X
LIST OF TABLES	XI
LIST OF ABBREVIATIONS	XII
1 CHAPTER I INTRODUCTION	1
1.1 HEVC Video Compression Standard.....	1
1.2 FVC Video Compression Standard.....	3
1.3 Thesis Contributions	4
1.4 Thesis Organization	6
2 CHAPTER II LOW COMPLEXITY HEVC SUB-PIXEL MOTION ESTIMATION TECHNIQUE AND ITS HARDWARE IMPLEMENTATION.....	7
2.1 HEVC Sub-Pixel Motion Estimation Algorithm	8
2.2 Proposed HEVC Sub-Pixel Motion Estimation Technique	9
2.3 Proposed HEVC Sub-Pixel Motion Estimation Hardware	10
3 CHAPTER III AN HEVC FRACTIONAL INTERPOLATION HARDWARE USING MEMORY BASED CONSTANT MULTIPLICATION	15
3.1 HEVC Fractional Interpolation Algorithm	16
3.2 Proposed HEVC Fractional Interpolation Hardware	18

3.3	Implementation Results.....	22
4	CHAPTER IV HIGH PERFORMANCE 2D TRANSFORM HARDWARE FOR FUTURE VIDEO CODING	25
4.1	FVC Transform Algorithms.....	27
4.2	Proposed FVC Baseline 2D Transform Hardware.....	29
4.3	Proposed FVC Reconfigurable 2D Transform Hardware.....	35
4.4	Proposed FVC Reconfigurable_DSP 2D Transform Hardware	38
4.5	Implementation Results.....	39
4.6	Implementation on FPGA Board	45
5	CHAPTER V CONCLUSIONS AND FUTURE WORK.....	48
6	BIBLIOGRAPHY	49

LIST OF FIGURES

Figure 1.1 HEVC Encoder Block Diagram.....	2
Figure 1.2 HEVC Decoder Block Diagram	2
Figure 2.1 Sub-pixel Search Locations	8
Figure 2.2 9x9 Integer Pixels	9
Figure 2.3 Proposed HEVC Sub-Pixel Motion Estimation Hardware	11
Figure 2.4 Type A, Type B and Type C FIR Filters	11
Figure 3.1 Integer, Half and Quarter Pixels	17
Figure 3.2 Type A, Type B and Type C FIR Filters	17
Figure 3.3 Proposed HEVC Fractional Interpolation Hardware	18
Figure 3.4 Multiplication Operations: (a) 5xA; (b) 17xA; (c) -11xA; (d) 29xA.....	20
Figure 3.5 MEM1 and MEM2.....	21
Figure 3.6 Energy Consumptions of FIHW_ORG, FIHW_MCM, FIHW_DSP and FIHW_MEM	23
Figure 4.1 Proposed FVC Baseline 2D Transform Hardware.....	30
Figure 4.2 1D DCT-II/DST-I Column Datapath.....	31
Figure 4.3 1D DCT-V/DCT-VIII/DST-VII Column Datapath	32
Figure 4.4 A Multiplier Block.....	33
Figure 4.5 Transpose Memory	34
Figure 4.6 Proposed FVC Reconfigurable 2D Transform Hardware.....	35
Figure 4.7 Reconfigurable 1D Column Datapath of the Proposed FVC Reconfigurable 2D Transform Hardware	36
Figure 4.8 Reconfigurable Multiplier Block	37
Figure 4.9 Reconfigurable 1D Column Datapath of the Proposed FVC Reconfigurable_DSP 2D Transform Hardware.....	38
Figure 4.10 Energy Consumption Results.....	43
Figure 4.11 Proposed FVC Reconfigurable 2D Transform Hardware Implementation on FPGA Board.....	47

LIST OF TABLES

Table 2.1 Computation Amount for Square-Shaped PU Sizes	10
Table 2.2 PSNR and SSIM Results.....	10
Table 2.3 Constant Coefficients	12
Table 2.4 Power Consumption Results	13
Table 2.5 Hardware Comparison	14
Table 3.1 Constant Coefficients	19
Table 3.2 Implementation Results.....	23
Table 3.3 Hardware Comparison	24
Table 4.1 DCT-II, DCT-V, DCT-VIII, DST-I, DST-VII Basis Functions	27
Table 4.2 Addition and Shift Amounts	29
Table 4.3 Transform Sets	29
Table 4.4 Adder and Shifter Amounts in 1D Datapaths.....	40
Table 4.5 Multiplier, Adder and Multiplexer Amounts in 1D Datapaths	41
Table 4.6 FPGA Implementation Results	41
Table 4.7 ASIC Implementation Results.....	42
Table 4.8 Comparison of FPGA Implementations	44
Table 4.9 Comparison of ASIC Implementations.....	45

LIST OF ABBREVIATIONS

AD	Absolute Difference
ALF	Adaptive Loop Filter
AMT	Adaptive Multiple Transform
ASIC	Application Specific Integrated Circuits
BRAM	Block Ram
CABAC	Context Adaptive Binary Arithmetic Coding
CU	Coding Unit
DBF	Deblocking Filter
DSP	Digital Signal Processor
DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
DDR RAM	Double Data Rate Ram
FPGA	Field Programmable Gate Array
FIR	Finite Impulse Response
FPS	Frame Per Second
FVC	Future Video Coding
HD	High Definition
HEVC	High Efficiency Video Coding
HM	HEVC Test Model
IDCT	Inverse Discrete Cosine Transform
IDST	Inverse Discrete Sine Transform
JEM	Joint Exploration Test Model
JCT-VC	Joint Collaborative Team on Video Coding
MV	Motion Vector
MCM	Multiple Constant Multiplication
PSNR	Peak Signal to Noise Ratio
PU	Prediction Unit
SAD	Sum of Absolute Differences

SPME	Sub-Pixel Motion Estimation
QP	Quantization Parameter
SAO	Sample Adaptive Offset
SSIM	Structural Similarity Index
TU	Transform Unit
UART	Universal Asynchronous Receiver/Transmitter
VCD	Value Change Dump

CHAPTER I

INTRODUCTION

1.1 HEVC Video Compression Standard

High Efficiency Video Coding (HEVC) is the current state-of-the-art video compression standard developed by Collaborative Team on Video Coding (JCT-VC) [1, 2, 3, 4]. HEVC provides 50% better coding efficiency than H.264 which is the previous video compression standard. HEVC also provides 23% bit rate reduction for the intra prediction only case [5, 6, 7]. HEVC standard achieves its video compression efficiency by combining a number of encoding tools such as intra prediction, inter prediction, transform, deblocking filter (DBF), sample adaptive offset (SAO) and entropy coder.

The top-level block diagrams of an HEVC encoder and decoder are shown in Figure 1.1 and Figure 1.2, respectively. An HEVC encoder has a forward path and a reconstruction path. The forward path is used to encode a video frame by using intra and inter predictions and to create the bit stream after the transform and quantization processes. Reconstruction path in the encoder ensures that both encoder and decoder use identical reference frames for intra and inter predictions. Since a decoder never gets original images, this avoids mismatch between encoder and decoder.

Reconstruction path begins with inverse quantization and inverse transform operations. The quantized transform coefficients are inverse quantized and inverse transformed to generate the reconstructed residual data. Since quantization is a lossy process, inverse quantized and inverse transformed coefficients are not identical to the original residual data. The reconstructed residual data are added to the predicted pixels in order to create the reconstructed frame. DBF is, then, applied to reduce the effects of blocking artifacts in the reconstructed frame.

HEVC intra prediction algorithm predicts the pixels of a block from the pixels of its already coded and reconstructed neighboring blocks. In HEVC standard, for the luminance component of a frame, intra PU sizes can be from 4x4 up to 32x32 and number of intra prediction modes for intra PU can be up to 35 [1, 8].

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 previous frames. In HEVC standard, inter PU sizes can be from 4x8/8x4 up to 64x64. HEVC inter prediction algorithm uses integer pixel motion estimation and sub-pixel (half and quarter) motion estimation operations. First, integer pixel motion estimation is performed for an inter PU. Then, sub-pixel (half and quarter) motion estimation is performed for the same inter PU. In HEVC, three different 8-tap FIR filters are used for both half-pixel and quarter-pixel interpolations [1, 2, 3, 4].

Integer based DCT is used in HEVC. TU sizes can be square-shaped sizes from 4x4 up to 32x32. In addition to DCT, HEVC uses DST for the 4x4 intra prediction case. Inverse discrete cosine transform (IDCT) and inverse discrete sine transform (IDST) are used in the reconstruction path of encoder and decoder [1, 2, 3, 7].

Entropy coder uses context adaptive binary arithmetic coding (CABAC) similar to H.264 with several improvements [2].

Deblocking filter algorithm reduces the blocking artifacts on the edges of prediction units. SAO and ALF are added to deblocking filter process in HEVC which are not used in previous video compression standards [1, 2, 3].

1.2 FVC Video Compression Standard

Since better coding efficiency is required for high resolution videos, JCT-VC is currently developing a new video compression standard called Future Video Coding (FVC) [9, 10]. FVC will offer much better compression efficiency than HEVC which is the current state-of-the-art video compression standard. FVC will have a similar top-

level block diagram with HEVC. But, algorithms used in each block will be improved for better compression efficiency at the expense of much more computational complexity.

FVC intra prediction algorithm performs the same operation as HEVC intra prediction algorithm. In FVC, number of directional intra prediction modes for an intra PU is increased from 33 to 65. Planar and DC intra prediction modes are the same as HEVC. In HEVC, 2-tap linear interpolation filter is used for directional intra prediction modes. 4-tap cubic and gaussian interpolation filters are used for directional intra prediction modes in FVC [8,10].

FVC inter prediction algorithm performs the same two-stage operation as HEVC. In HEVC 1/4, one-quarter, motion vector accuracy is used. In FVC, 1/16 motion vector accuracy is added for merge/skip modes. In FVC, motion vector prediction process used in HEVC is improved for better compression efficiency [1, 2, 10].

Integer based DCT is used in FVC same as HEVC. HEVC transform algorithm uses DCT-II. It also uses DST-VII for the 4x4 intra prediction case. In HEVC, TU sizes can be from 4x4 up to 32x32. [1, 2]. In FVC transform algorithm, an Adaptive Multiple Transform (AMT) scheme is used. AMT scheme uses DCT-II, DCT-V, DCT-VIII, DST-I and DST-VII based on prediction (intra or inter) type. In FVC, TU sizes can be from 4x4 up to 64x64. Mode dependent non-separable secondary transform and signal dependent transform are also added to FVC [9, 10, 11, 12].

Entropy coder uses CABAC similar to HEVC with several enhancements. Deblocking filter algorithm in FVC is the same as HEVC [1, 2, 10].

1.3 Thesis Contributions

We propose a low complexity sub-pixel motion estimation (SPME) technique [13]. In HEVC, SPME is performed to obtain sub-pixel accurate motion vector (MV) after integer pixel motion estimation. SPME first interpolates necessary sub-pixels for sub-pixel search locations. Then, it calculates the sum of absolute difference (SAD) values for each sub-pixel search location and determines the best sub-pixel search location with the minimum SAD. SPME has high computational complexity due to these operations. Therefore, we propose interpolating SAD values of sub-pixel search locations using the SAD values of neighboring integer pixel search locations instead of interpolating necessary sub-pixels and calculating SAD values for sub-pixel search

locations. In this way, number of interpolation operation is significantly reduced and absolute difference (AD) operation is not required with a slight decrease in PSNR.

We also implemented a high performance HEVC SPME hardware implementing the proposed technique for all PU sizes using Verilog HDL [13]. We mapped the Verilog RTL code to a Xilinx Virtex 6 FPGA. The proposed hardware, in the worst case, can process 38 quad full HD (QFHD) (3840x2160) video frames per second.

We designed an HEVC fractional (half-pixel and quarter-pixel) interpolation hardware using memory based constant multiplication for all PU sizes. The proposed hardware uses memory based constant multiplication technique for implementing multiplications with constant coefficients. The proposed memory based constant multiplication hardware stores pre-computed products of an input pixel with multiple constant coefficients in memory. Several optimizations are proposed to reduce memory size. The proposed hardware is implemented using Verilog HDL. We mapped the Verilog RTL code to a Xilinx Virtex 6 FPGA and estimated its energy consumption on this FPGA using Xilinx XPower Analyzer tool. The proposed HEVC fractional interpolation hardware using memory based constant multiplication has up to 31% less energy consumption than original HEVC fractional interpolation hardware. The proposed HEVC fractional interpolation hardware using memory based constant multiplication has up to 12.3% and 4.4% less energy consumption than HEVC fractional interpolation hardware implementing constant coefficient multiplications using Hcub multiplierless constant multiplication (MCM) algorithm and DSP blocks in Xilinx Virtex-6 FPGA, respectively. The proposed hardware, in the worst case, can process 35 QFHD (3840x2160) video frames per second.

HEVC transform algorithm uses DCT-II and DST-VII. FVC transform algorithm uses DCT-II, DCT-V, DCT-VIII, DST-I and DST-VII in order to increase compression efficiency at the expense of higher computational complexity. In this thesis, we designed three different high performance FVC 2D transform hardware for 4x4 and 8x8 TU sizes [14], [15]. The proposed hardware are implemented using Verilog HDL. We mapped the Verilog RTL codes to a Xilinx Virtex 6 FPGA and estimated their power consumptions on this FPGA using Xilinx XPower Analyzer tool.

The first proposed hardware (baseline) uses separate datapaths for each 1D transform and it uses Hcub MCM algorithm for implementing 1D transforms. It uses data gating technique and the data gating technique reduced the energy consumption of the proposed baseline hardware up to 71.7%. The second proposed hardware

(reconfigurable) uses one reconfigurable datapath for all 1D column transforms and one reconfigurable datapath for all 1D row transforms. The proposed reconfigurability reduced hardware area at the expense of energy consumption increase. Therefore, the baseline hardware can be used in high performance and low energy FVC encoders. The reconfigurable hardware can be used in high performance and low cost FVC encoders.

The third proposed hardware (reconfigurable_DSP) uses one reconfigurable datapath for all 1D column transforms and one reconfigurable datapath for all 1D row transforms. It uses built-in full-custom DSP blocks in Xilinx Virtex-6 FPGA for implementing 1D transforms. Since it is more efficient to implement constant multiplications using DSP blocks in an FPGA implementation, FPGA implementation of the reconfigurable_DSP hardware has up to 29% and 59% less energy consumption than FPGA implementations of the baseline and reconfigurable hardware, respectively.

1.4 Thesis Organization

The rest of the thesis is organized as follows.

Chapter II explains HEVC sub-pixel motion estimation algorithm. It presents the proposed low complexity HEVC sub-pixel motion estimation technique. It describes the proposed high performance HEVC sub-pixel motion estimation hardware implementing the proposed technique and presents its implementation results.

Chapter III explains HEVC fractional interpolation algorithm. It describes the proposed HEVC fractional interpolation hardware using memory based constant multiplication and presents its implementation results.

Chapter IV presents FVC transform algorithm used in FVC encoder. It presents three different proposed high performance FVC 2D transform hardware and their implementation results.

Chapter V presents conclusions and future work.

CHAPTER II

LOW COMPLEXITY HEVC SUB-PIXEL MOTION ESTIMATION TECHNIQUE AND ITS HARDWARE IMPLEMENTATION

In order to increase the performance of integer pixel motion estimation, SPME, which provides sub-pixel accurate MV refinement, is performed. HEVC uses SPME same as H.264. However, HEVC SPME has higher computational complexity than H.264 SPME. HEVC standard uses three different 8-tap FIR filters for sub-pixel interpolation and up to 64x64 PU sizes [16, 17]. SPME is heavily used in an HEVC encoder [5]. It accounts for up to 49% of total encoding time of HEVC video encoder.

In this thesis, a low complexity HEVC SPME technique for all PU sizes is proposed. The proposed technique interpolates the SAD values of sub-pixel search locations using the SAD values of neighboring integer pixel search locations. In this thesis, an efficient HEVC SPME hardware implementing the proposed technique for all PU sizes is also designed and implemented using Verilog HDL. In order to reduce number and size of adders in this hardware, Hcub MCM algorithm is used [18]. The proposed hardware finishes SPME for a PU in 6 clock cycles. It, in the worst case, can process 38 QFHD (3840x2160) video frames per second.

Several HEVC SPME hardware are proposed in the literature [19, 20, 21]. In [19], SPME hardware searches all possible 48 sub-pixel search locations. However, it only supports square shaped PU sizes. In [20], SPME hardware supports all PU sizes but 8x4, 4x8 and 8x8. It uses bilinear filter for quarter-pixel interpolation. Also, it searches 12 sub-pixel search locations. In [21], SPME hardware supports all PU sizes

but it uses a scalable search pattern. HEVC SPME hardware proposed in this thesis is compared with these HEVC SPME hardware.

2.1 HEVC Sub-Pixel Motion Estimation Algorithm

After integer pixel motion estimation is performed for a PU, SPME is performed for the same PU to obtain sub-pixel accurate MV. In HEVC Test Model (HM) reference software video encoder [22], SPME is performed in two stages. As shown in Figure 2.1, 8 sub-pixel search locations around the best integer pixel search location are searched in the first stage. 8 sub-pixel search locations around the best sub-pixel search location of the first stage are searched in the second stage.

HEVC SPME first interpolates the necessary sub-pixels for sub-pixel search locations using three different 8-tap FIR filters. In Figure 2.1, half-pixels a, b, c and d, h, n are interpolated using the nearest integer pixels in horizontal and vertical directions, respectively. Quarter-pixels e, i, p and f, j, q and g, k, r are interpolated using the nearest a and b and c half-pixels in vertical directions, respectively. HEVC SPME then calculates the SAD values for each sub-pixel search location, and determines the best sub-pixel search location with the minimum SAD value.

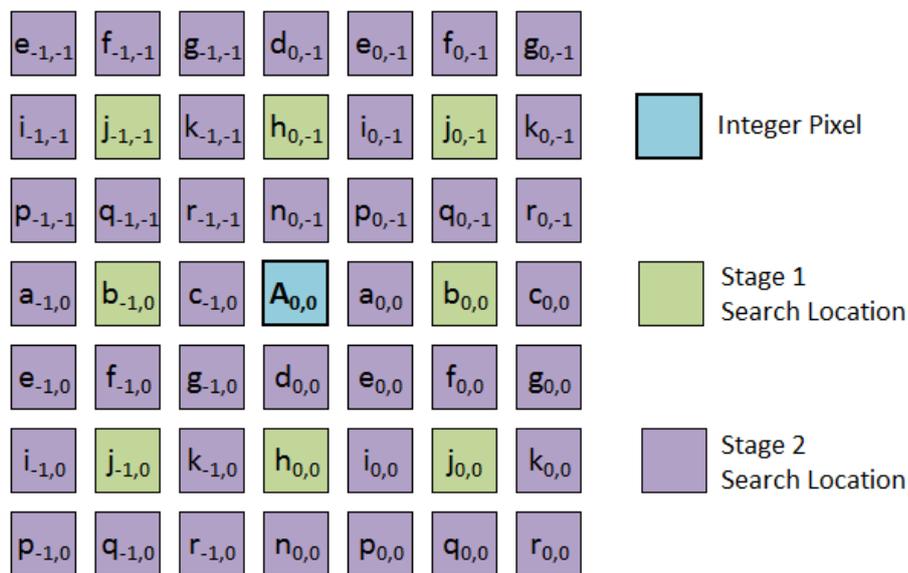


Figure 2.1 Sub-pixel Search Locations

2.2 Proposed HEVC Sub-Pixel Motion Estimation Technique

The proposed HEVC SPME technique interpolates SAD values of sub-pixel search locations using the SAD values of neighboring integer pixel search locations. As shown in Figure 2.2, the proposed technique uses SAD values of the best integer pixel search location, $A_{0,0}$, and its neighboring 80 integer pixel search locations, a 9x9 SAD block, for directly interpolating SAD values of 48 sub-pixel search locations using HEVC sub-pixel interpolation FIR filters. SAD values of half-pixel search locations are interpolated using the SAD values of nearest integer pixel search locations. SAD values of quarter-pixel search locations are interpolated using the SAD values of a, b, c half-pixel search locations.

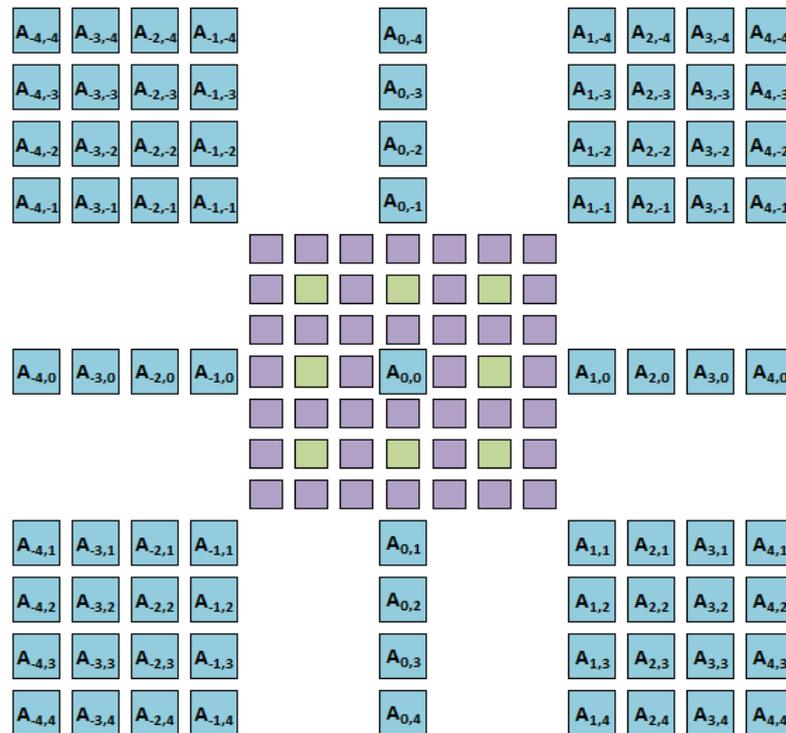


Figure 2.2 9x9 Integer Pixels

The proposed technique performs SPME in two stages, same as HM reference software video encoder [22]. However, it performs SPME without interpolating a sub-pixel and calculating an AD. Table 2.1 shows the number of interpolation and AD operations required for performing HEVC SPME for one square-shaped PU. Since the proposed technique only interpolates SAD values of sub-pixel search locations, number of interpolation operations is significantly reduced and AD operation is not required.

Table 2.1 Computation Amount for Square-Shaped PU Sizes

PU Sizes	Original HEVC SPME				Proposed
	8x8	16x16	32x32	64x64	
Number of Interpolations	1377	4641	16929	64545	100
Number of Abs. Diff.	1024	4096	16384	65536	0

The proposed HEVC SPME technique is implemented in MATLAB. As shown in Table 2.2, MATLAB simulation results show that it slightly decreases PSNR and achieves good structural similarity index (SSIM) results.

Table 2.2 PSNR and SSIM Results

	Frame	Δ PSNR (dB)	SSIM
Class B (1920x1080)	Tennis	-0.847	0.975
	Kimono	-0.225	0.982
	Basketball D.	-0.015	0.970
	Park Scene	-0.313	0.974

2.3 Proposed HEVC Sub-Pixel Motion Estimation Hardware

The proposed HEVC SPME hardware for all PU sizes is shown in Figure 2.3. It takes 9x9 20-bit SAD values of 9x9 integer pixel search locations as input into integer SAD buffer. Three buffers are used to store the SAD values of sub-pixel search locations. These on-chip buffers reduce the required off-chip memory bandwidth and power consumption.

The proposed hardware has three interpolation units. Each interpolation unit takes 9 SAD values as input and interpolates 20-bit SAD values of $3 \times 2 = 6$ sub-pixel search locations in each clock cycle. It interpolates 2 SAD values using type A, 2 SAD values using type B and 2 SAD values using type C FIR filter equations. As shown in Figure 2.4, common expressions are calculated in type A, type B and type C FIR filter equations and same integer pixel is multiplied with different constant coefficients in type A, type B and type C FIR filter equations. Therefore, in an interpolation unit, common expressions in different equations are calculated once, and the results are used in all the equations [17].

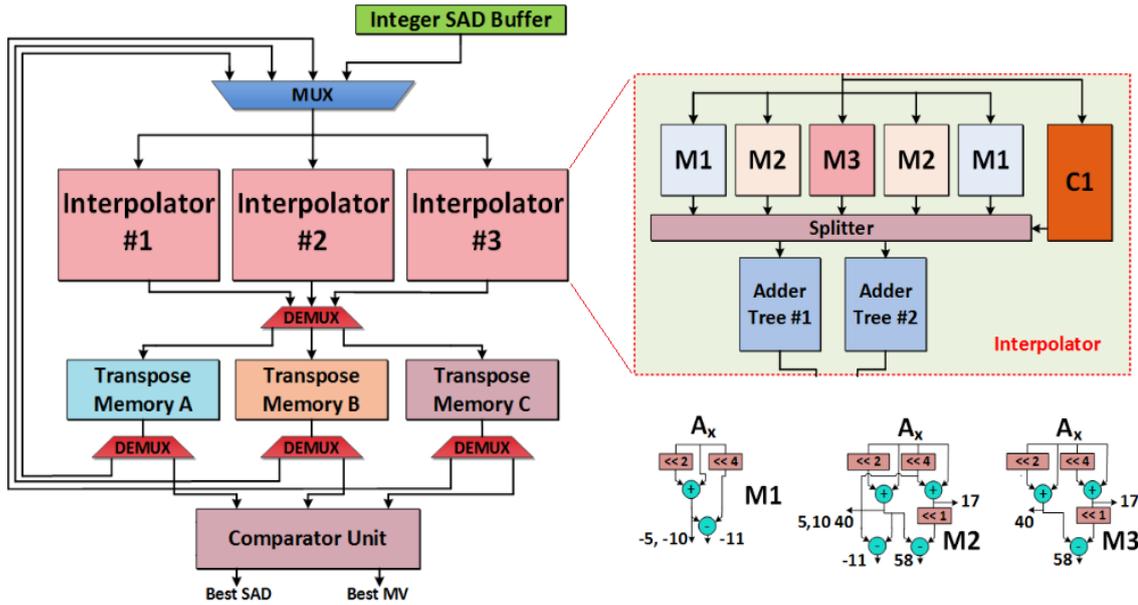


Figure 2.3 Proposed HEVC Sub-Pixel Motion Estimation Hardware

Multiplications in FIR filter equations are performed using only adders and shifters. In the proposed hardware, Hcub MCM algorithm is used to reduce number and size of the adders, and to minimize adder tree depth [18]. Hcub algorithm tries to minimize number of adders, their bit size and adder tree depth in a multiplier block, which multiplies a single input with multiple constants. A multiplier block hardware has only one input, and it outputs results of multiplications with all the constants. Hcub algorithm determines necessary shift and addition operations in a multiplier block.

$$\begin{aligned}
 a_{-1,0} &= -A_{-4} + 4xA_{-3} - 10xA_{-2} + 58xA_{-1} + 17xA_0 - 5xA_1 + A_2 \\
 a_{0,0} &= -A_{-3} + 4xA_{-2} - 10xA_{-1} + 58xA_0 + 17xA_1 - 5xA_2 + A_3 \\
 b_{-1,0} &= -A_{-4} + 4xA_{-3} - 11xA_{-2} + 40xA_{-1} + 40xA_0 - 11xA_1 + 4xA_2 - A_3 \\
 b_{0,0} &= -A_{-3} + 4xA_{-2} - 11xA_{-1} + 40xA_0 + 40xA_1 - 11xA_2 + 4xA_3 - A_4 \\
 c_{-1,0} &= A_{-3} - 5xA_{-2} + 17xA_{-1} + 58xA_0 - 10xA_1 + 4xA_2 - A_3 \\
 c_{0,0} &= A_{-2} - 5xA_{-1} + 17xA_0 + 58xA_1 - 10xA_2 + 4xA_3 - A_4
 \end{aligned}$$

Figure 2.4 Type A, Type B and Type C FIR Filters

As shown in Table 2.3, since different constant coefficients are used in FIR filter equations, three different multiplier blocks are used. Common 1 (C1) datapath calculates the common sub-expressions in the equations shown in the blue boxes in Figure 2.4. Multiplier 1 (M1), Multiplier 2 (M2), and Multiplier 3 (M3) datapaths

calculate the multiplications with multiple constant coefficients for different set of coefficients. For example, M2 datapath calculates the multiplications for A_1 written with red color in Figure 2.4.

Table 2.3 Constant Coefficients

Input SADs	Coefficients	Datapath
A_4	-1	C1
A_3	-1, 4	
A_2	4, -5, -10, -11	M1
A_1	-5, -10, -11, 17, 40, 58	M2
A_0	17, 58, 40	M3
A_1	-5, -10, -11, 17, 40, 58	M2
A_2	4, -5, -10, -11	M1
A_3	-1, 4	C1
A_4	-1	

Comparator unit compares the SAD values of sub-pixel search locations, and determines the best sub-pixel search location with minimum SAD value. It uses three 20-bit comparators and performs comparison in 6 clock cycles.

SAD values of 48 sub-pixel search locations should be interpolated. First, 9x2 SAD values of a, b, c half-pixel search locations necessary for interpolating SAD values of quarter-pixel search locations are interpolated using SAD values of integer pixel search locations in 3 clock cycles. Then, 2x1 SAD values of d, h, n half-pixel search locations are interpolated using SAD values of integer pixel search locations in 1 clock cycle. Finally, 2x2 SAD values of quarter-pixel search locations are interpolated using SAD values of a, b, c half-pixel search locations in 2 clock cycles.

Because of the input data loading and pipelining, the proposed hardware starts producing outputs after 12 clock cycles. It then continues producing outputs at every 6 clock cycles without any stall. Therefore, it finishes SPME for a PU in 6 clock cycles.

The proposed HEVC SPME hardware for all PU sizes including the proposed technique is implemented using Verilog HDL. The Verilog RTL implementation is verified with RTL simulations. RTL simulation results matched the results of MATLAB implementation of HEVC SPME including the proposed technique.

The Verilog RTL code is synthesized and mapped to a XC6VLX365T Xilinx Virtex 6 FPGA with speed grade 3. The FPGA implementation is verified with post

place & route simulations. The FPGA implementation uses 5200 LUTs, 1814 Slices and 3794 DFFs. The FPGA implementation works at 142 MHz. It can process 19 QFHD (3840x2160) video frames per second.

Power consumption of the FPGA implementation is estimated using Xilinx XPower Analyzer tool. Post place & route timing simulations are performed for Tennis, Kimono, BQ Terrace and Basketball Drive class B videos (one frame from each video) at 100 MHz [23] and signal activities are stored in VCD files. These VCD files are used for estimating power consumption of the FPGA implementation. These power consumption results are shown in Table 2.4.

Table 2.4 Power Consumption Results

	Tennis	Kimono	BQ Terr.	B. Drive
Clock (mW)	33	33	33	33
Logic (mW)	68	79	78	67
Signal (mW)	143	168	163	139
Total Power (mW)	244	280	274	239

In order to compare the proposed HEVC SPME hardware with the HEVC SPME hardware in the literature, Verilog RTL code is also synthesized to a 90 nm standard cell library and resulting netlist is placed and routed. The resulting ASIC implementation works at 280 MHz. It can process 38 QFHD (3840x2160) video frames per second. Gate count of the ASIC implementation is calculated as 26K according to NAND (2x1) gate area excluding on-chip memory.

The comparison of the proposed HEVC SPME hardware with the HEVC SPME hardware in the literature is shown in Table 2.5. The proposed hardware implements HEVC SPME for all PU sizes and it is the only hardware that implements the two stages SPME performed in HM reference software video encoder [22]. It has higher throughput, and it has smaller area and lower power consumption than the other HEVC SPME hardware. HEVC SPME hardware proposed in [21] has higher throughput than FPGA implementation of the proposed hardware. However, it has 70 times larger area than FPGA implementation of the proposed hardware.

Table 2.5 Hardware Comparison

	[19]	[20]	[21]	Proposed	
Technology	65 nm	65 nm	Xilinx Virtex6	90 nm	Xilinx Virtex6
Gate/Slices	249.1 K	1183 K	130306	26 K	1814
Max Freq. (MHz)	396.8	188	200	280	142
Power Dissip. (mW)	48.67	198.6	---	28	280
Supported PU sizes	Square Shaped	All but 8x8, 8x4 and 4x8	All	All	All
Fps	60 QFHD	30 QFHD	32 QFHD	38 QFHD	19 QFHD
Fps * (Normalized)	6 QFHD	15 QFHD	32 QFHD	38 QFHD	19 QFHD

*: Frames per second when hardware processes all PU sizes

CHAPTER III

AN HEVC FRACTIONAL INTERPOLATION HARDWARE USING MEMORY BASED CONSTANT MULTIPLICATION

Fractional (half-pixel and quarter-pixel) interpolation is one of the most computationally intensive parts of HEVC video encoder and decoder. Fractional interpolation operation accounts for 25% and 50% of the HEVC encoder and decoder complexity, respectively [5].

In H.264 standard, a 6-tap FIR filter is used for half-pixel interpolation and bilinear filter is used for quarter-pixel interpolation [1, 7]. In HEVC standard, three different 8-tap FIR filters are used for half-pixel and quarter-pixel interpolations. Block sizes from 4x4 to 16x16 are used in H.264 standard. However, in HEVC standard, PU sizes can be from 4x8/8x4 to 64x64. Therefore, HEVC fractional interpolation is more complex than H.264 fractional interpolation.

Memory based constant multiplication is an efficient computation technique [24, 25]. A memory based constant multiplication hardware stores pre-computed product values for an input word into memory and necessary product value is read from the memory using input word as the address.

In this thesis, an HEVC fractional interpolation hardware using memory based constant multiplication for all PU sizes is designed and implemented using Verilog HDL. The proposed hardware uses memory based constant multiplication technique for implementing multiplication with constant coefficients. The proposed memory based constant multiplication hardware stores pre-computed products of an input pixel with

multiple constant coefficients in memory. Several optimizations are proposed to reduce memory size.

Several HEVC fractional interpolation hardware are proposed in the literature [16, 17, 26, 27, 28]. In Section 3.3, they are compared with HEVC fractional interpolation hardware proposed in this thesis. They do not use memory based constant multiplication technique.

In [16], three different 8-tap FIR filters are implemented using a reconfigurable datapath. It can calculate one FIR filter output at a time. Therefore, it can only be used for motion compensation. The proposed hardware in [17] uses Hcub MCM algorithm for multiplication with constant coefficients. In [26, 27, 28], the proposed hardware use adders and shifters for FIR filter implementation.

3.1 HEVC Fractional Interpolation Algorithm

In HEVC, three different 8-tap FIR filters are used for both half-pixel and quarter-pixel interpolations. These three FIR filters type A, type B and type C are shown in (3.1), (3.2), and (3.3), respectively. The *shift1* value is determined based on bit depth of the pixel [1, 4].

$$a_{0,0} = (-A_{-3,0} + 4 * A_{-2,0} - 10 * A_{-1,0} + 58 * A_{0,0} + 17 * A_{1,0} - 5 * A_{2,0} + A_{3,0}) \gg \text{shift1} \quad (3.1)$$

$$b_{0,0} = (-A_{-3,0} + 4 * A_{-2,0} - 11 * A_{-1,0} + 40 * A_{0,0} + 40 * A_{1,0} - 11 * A_{2,0} + 4 * A_{3,0} - A_{4,0}) \gg \text{shift1} \quad (3.2)$$

$$c_{0,0} = (A_{-2,0} - 5 * A_{-1,0} + 17 * A_{0,0} + 58 * A_{1,0} - 10 * A_{2,0} + 4 * A_{3,0} - A_{4,0}) \gg \text{shift1} \quad (3.3)$$

Integer pixels ($A_{x,y}$), half pixels ($a_{x,y}$, $b_{x,y}$, $c_{x,y}$, $d_{x,y}$, $h_{x,y}$, $n_{x,y}$) and quarter pixels ($e_{x,y}$, $f_{x,y}$, $g_{x,y}$, $i_{x,y}$, $j_{x,y}$, $k_{x,y}$, $p_{x,y}$, $q_{x,y}$, $r_{x,y}$) in a PU are shown in Figure 3.1. The type A, type B and type C FIR filter equations for 8 half-pixels are shown in Figure 3.2.

The half pixels a, b, c are interpolated from nearest integer pixels in horizontal direction, and the half-pixels d, h, n are interpolated from nearest integer pixels in vertical direction. The quarter pixels e, f, g are interpolated from the nearest half pixels a, b, c, respectively, in vertical direction using type A filter. The quarter pixels i, j, k are interpolated similarly using type B filter, and the quarter pixels p, q, r are interpolated similarly using type C filter.

$A_{-1,-1}$				$A_{0,-1}$	$a_{0,-1}$	$b_{0,-1}$	$c_{0,-1}$	$A_{1,-1}$				$A_{2,-1}$
$A_{-1,0}$				$A_{0,0}$	$a_{0,0}$	$b_{0,0}$	$c_{0,0}$	$A_{1,0}$				$A_{2,0}$
$d_{-1,0}$				$d_{0,0}$	$e_{0,0}$	$f_{0,0}$	$g_{0,0}$	$d_{1,0}$				$d_{2,0}$
$h_{-1,0}$				$h_{0,0}$	$i_{0,0}$	$j_{0,0}$	$k_{0,0}$	$h_{1,0}$				$h_{2,0}$
$n_{-1,0}$				$n_{0,0}$	$p_{0,0}$	$q_{0,0}$	$r_{0,0}$	$n_{1,0}$				$n_{2,0}$
$A_{-1,1}$				$A_{0,1}$	$a_{0,1}$	$b_{0,1}$	$c_{0,1}$	$A_{1,1}$				$A_{2,1}$
$A_{-1,1}$				$A_{0,1}$	$a_{0,2}$	$b_{0,2}$	$c_{0,2}$	$A_{1,1}$				$A_{2,1}$

Figure 3.1 Integer, Half and Quarter Pixels

HEVC fractional interpolation algorithm used in HEVC encoder calculates all the fractional pixels necessary for the fractional motion estimation operation.

$$\begin{aligned}
 a_{-3,0} &= -A_{-6} + 4xA_{-5} - 10xA_{-4} + 58xA_{-3} + 17xA_{-2} - 5xA_{-1} + A_0 \\
 a_{-2,0} &= -A_{-5} + 4xA_{-4} - 10xA_{-3} + 58xA_{-2} + 17xA_{-1} - 5xA_0 + A_1 \\
 a_{-1,0} &= -A_{-4} + 4xA_{-3} - 10xA_{-2} + 58xA_{-1} + 17xA_0 - 5xA_1 + A_2 \\
 a_{0,0} &= -A_{-3} + 4xA_{-2} - 10xA_{-1} + 58xA_0 + 17xA_1 - 5xA_2 + A_3 \\
 a_{1,0} &= -A_{-2} + 4xA_{-1} - 10xA_0 + 58xA_1 + 17xA_2 - 5xA_3 + A_4 \\
 a_{2,0} &= -A_{-1} + 4xA_0 - 10xA_1 + 58xA_2 + 17xA_3 - 5xA_4 + A_5 \\
 a_{3,0} &= -A_0 + 4xA_1 - 10xA_2 + 58xA_3 + 17xA_4 - 5xA_5 + A_6 \\
 a_{4,0} &= -A_1 + 4xA_2 - 10xA_3 + 58xA_4 + 17xA_5 - 5xA_6 + A_7
 \end{aligned}$$

$$\begin{aligned}
 b_{-3,0} &= -A_{-6} + 4xA_{-5} - 11xA_{-4} + 40xA_{-3} + 40xA_{-2} - 11xA_{-1} + 4xA_0 - A_1 \\
 b_{-2,0} &= -A_{-5} + 4xA_{-4} - 11xA_{-3} + 40xA_{-2} + 40xA_{-1} - 11xA_0 + 4xA_1 - A_2 \\
 b_{-1,0} &= -A_{-4} + 4xA_{-3} - 11xA_{-2} + 40xA_{-1} + 40xA_0 - 11xA_1 + 4xA_2 - A_3 \\
 b_{0,0} &= -A_{-3} + 4xA_{-2} - 11xA_{-1} + 40xA_0 + 40xA_1 - 11xA_2 + 4xA_3 - A_4 \\
 b_{1,0} &= -A_{-2} + 4xA_{-1} - 11xA_0 + 40xA_1 + 40xA_2 - 11xA_3 + 4xA_4 - A_5 \\
 b_{2,0} &= -A_{-1} + 4xA_0 - 11xA_1 + 40xA_2 + 40xA_3 - 11xA_4 + 4xA_5 - A_6 \\
 b_{3,0} &= -A_0 + 4xA_1 - 11xA_2 + 40xA_3 + 40xA_4 - 11xA_5 + 4xA_6 - A_7 \\
 b_{4,0} &= -A_1 + 4xA_2 - 11xA_3 + 40xA_4 + 40xA_5 - 11xA_6 + 4xA_7 - A_8
 \end{aligned}$$

$$\begin{aligned}
 c_{-3,0} &= A_{-5} - 5xA_{-4} + 17xA_{-3} + 58xA_{-2} - 10xA_{-1} + 4xA_0 - A_1 \\
 c_{-2,0} &= A_{-4} - 5xA_{-3} + 17xA_{-2} + 58xA_{-1} - 10xA_0 + 4xA_1 - A_2 \\
 c_{-1,0} &= A_{-3} - 5xA_{-2} + 17xA_{-1} + 58xA_0 - 10xA_1 + 4xA_2 - A_3 \\
 c_{0,0} &= A_{-2} - 5xA_{-1} + 17xA_0 + 58xA_1 - 10xA_2 + 4xA_3 - A_4 \\
 c_{1,0} &= A_{-1} - 5xA_0 + 17xA_1 + 58xA_2 - 10xA_3 + 4xA_4 - A_5 \\
 c_{2,0} &= A_0 - 5xA_1 + 17xA_2 + 58xA_3 - 10xA_4 + 4xA_5 - A_6 \\
 c_{3,0} &= A_1 - 5xA_2 + 17xA_3 + 58xA_4 - 10xA_5 + 4xA_6 - A_7 \\
 c_{4,0} &= A_2 - 5xA_3 + 17xA_4 + 58xA_5 - 10xA_6 + 4xA_7 - A_8
 \end{aligned}$$

Figure 3.2 Type A, Type B and Type C FIR Filters

3.2 Proposed HEVC Fractional Interpolation Hardware

The proposed HEVC fractional interpolation hardware for all PU sizes is shown in Figure 3.2. The proposed hardware interpolates all the fractional pixels (half-pixels and quarter-pixels) for the luma component of a PU using integer or half-pixels. The proposed hardware is designed for 8x8 PU size and it produces necessary fractional pixels for an 8x8 PU. For other PU sizes, the PU is divided into 8x8 blocks, and the blocks are interpolated separately. For example, a 16x16 PU is divided into four 8x8 blocks and each 8x8 block is interpolated separately.

In the proposed hardware, 8x3 fractional pixels are interpolated in parallel using type A, type B and type C FIR filters. In the proposed hardware, common sub-expression calculation method proposed in [17] is used. As shown in Figure 3.3, there are common sub-expressions in different filter type equations. Common sub-expressions in type A and type B filters are shown in blue boxes. Common sub-expressions in type B and type C filters are shown in green boxes. In the proposed hardware, common sub-expressions in different equations are calculated once, and the results are used in all the equations. The common sub-expressions are calculated in CSE datapath using adders and shifters.

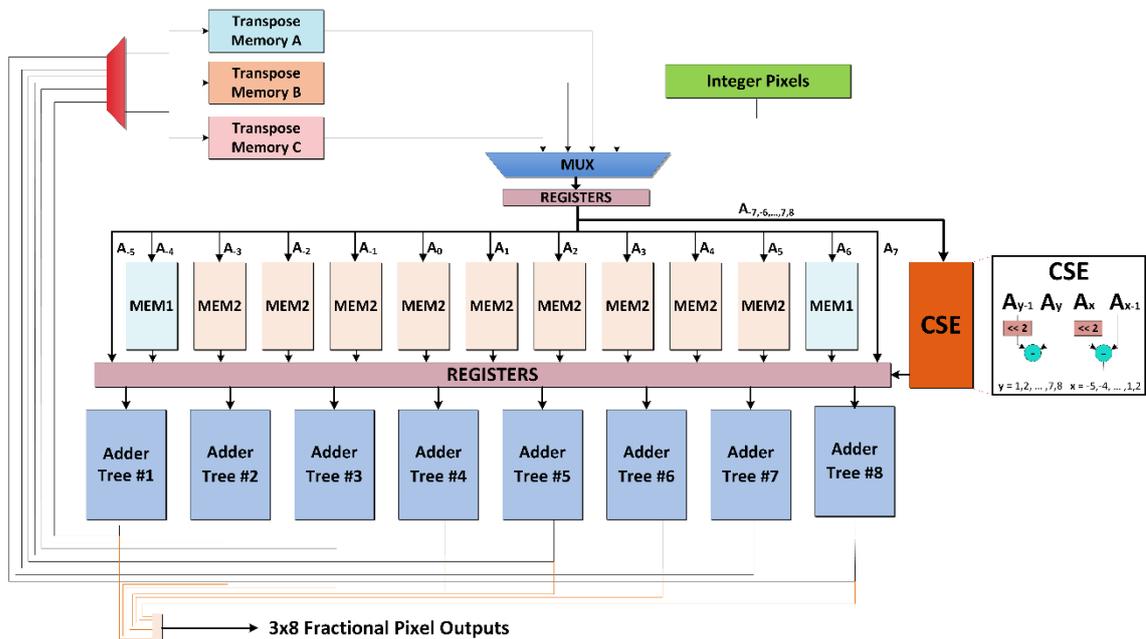


Figure 3.3 Proposed HEVC Fractional Interpolation Hardware

Three on-chip transpose memories are used to store half-pixels necessary for interpolating quarter-pixels. The half-pixels are interpolated using integer pixels and the interpolated a, b and c half-pixels are stored in the transpose memories A, B and C, respectively. These on-chip buffers reduce the required off-chip memory bandwidth and power consumption.

Each input pixel should be multiplied with multiple constant coefficients shown as red boxes in Figure 3.2. Table 3.1 shows constant coefficient multiplications necessary for each input pixel. In the proposed hardware, constant coefficient multiplications are implemented using memory based constant multiplication technique. As shown in Table 3.1, since constant coefficients of input pixels (A_4, A_6) and ($A_3 \dots A_5$) are different, two different memories, MEM1 and MEM2, are used to store pre-computed products of an input pixel with multiple constant coefficients.

Input pixels (A_4, A_6) need to be multiplied with constant coefficients 1, -5, -10 and -11. In the proposed hardware, MEM1 stores two product values $5xA$ and $-11xA$ for input pixel A. The product value $10xA$ is obtained from $5xA$ using shift operation. Input pixels ($A_3 \dots A_5$) need to be multiplied with constant coefficients 1, -5, -10, -11, 17, 40 and 58. In the proposed hardware, MEM2 stores four product values $5xA$, $-11xA$, $17xA$ and $29xA$ for input pixel A. Product values $10xA$ and $40xA$ are obtained from $5xA$ using shift operation. After constant coefficient multiplications are performed by memory based constant multiplication technique, fractional pixels are calculated using adder trees.

Table 3.1 Constant Coefficients

Input Pixel	Necessary Coefficients	Hardware	Stored Products
A_5	1	---	---
A_4	1,-5,-10,-11	MEM1	5,-11
A_3	1,-5,-10,-11,17,40,58		5,-11,17,29
A_2	1,-5,-10,-11,17,40,58		5,-11,17,29
A_1	1,-5,-10,-11,17,40,58		5,-11,17,29
A_0	1,-5,-10,-11,17,40,58		5,-11,17,29
A_1	1,-5,-10,-11,17,40,58	MEM2	5,-11,17,29
A_2	1,-5,-10,-11,17,40,58		5,-11,17,29
A_3	1,-5,-10,-11,17,40,58		5,-11,17,29
A_4	1,-5,-10,-11,17,40,58		5,-11,17,29
A_5	1,-5,-10,-11,17,40,58		5,-11,17,29
A_6	1,-5,-10,-11	MEM1	5,-11
A_7	1	---	---

8 bit unsigned input pixel A is used as the address of MEM1 and MEM2 memories. MEM1 stores 2 product values, $5xA$ and $-11xA$, in each address. MEM2 stores 4 product values, $5xA$, $-11xA$, $17xA$ and $29xA$, in each address. Since address ports of MEM1 and MEM2 are 8-bits, MEM1 and MEM2 store $2^8 \times 2$ and $2^8 \times 4$ product values, respectively.

Multiplications of an input pixel A with constant coefficients 5, -11, 17 and 29 using additions and shifts are shown in (3.4-3.7) and Figure 3.4. Products of an 8-bit unsigned input pixel with constant coefficients 5, -11, 17 and 29 are 11-bits, 13-bits, 13-bits and 13-bits, respectively. Therefore, MEM1 and MEM2 should store $11+13=24$ and $11+13+13+13=50$ bits in each address, respectively.

$$5xA = (A \ll 2) + A \quad (3.4)$$

$$-11xA = 5xA + ((A' + 1) \ll 4) \quad (3.5)$$

$$17xA = (A \ll 4) + A \quad (3.6)$$

$$29xA = (A \ll 4) + (A \ll 3) + 5xA \quad (3.7)$$

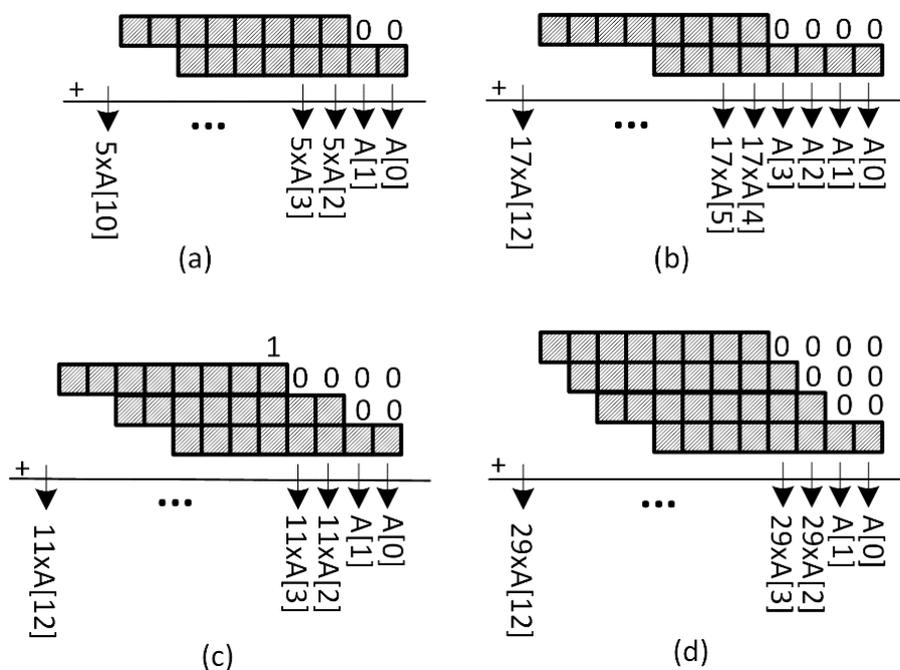


Figure 3.4 Multiplication Operations: (a) $5xA$; (b) $17xA$; (c) $-11xA$; (d) $29xA$.

As shown in Figure 3.4, least significant 2-bits of $5xA$, $-11xA$ and $29xA$, and least significant 4-bits of $17xA$ are equal to the bits of input pixel A . Therefore, these bits of the products do not need to be stored in memories. This optimization saves $2+2=4$ bits and $2+2+2+4=10$ bits in each address of MEM1 and MEM2, respectively. Also, least significant third bit of $A \times 5$ is equal to the least significant third bit of $-11xA$ and $29xA$, and the least significant fourth bit of $5xA$ is equal to the least significant fourth bit of $-11xA$. Therefore, only least significant third and fourth bits of $5xA$ need to be stored in memories and they should be used for $5xA$, $-11xA$ and $29xA$. This optimization saves 2 bits and $2+1=3$ bits in each address of MEM1 and MEM2, respectively.

Using these optimizations, number of bits in each address of MEM1 is reduced from 24 to 18 and number of bits in each address of MEM2 is reduced from 50 to 37. The proposed memories, MEM1 and MEM2, are shown in Figure 3.5.

Since 15 fractional pixels should be interpolated for each integer pixel, 64×15 fractional pixels should be interpolated for an 8×8 PU. 8×7 extra a, b, c half-pixels are necessary for the interpolation of quarter-pixels.

First, 8×15 a, b and c half-pixels necessary for interpolating quarter-pixels are interpolated in 15 clock cycles, and stored in the transpose memories A, B and C, respectively. Then, 8×8 d, h, n half-pixels are interpolated in 8 clock cycles. Finally, $9 \times 8 \times 8$ quarter-pixels are interpolated in 8×3 clock cycles using a, b and c half-pixels. There are three pipeline stages in the proposed hardware. Therefore, the proposed hardware, in the worst case, interpolates the fractional pixels for an 8×8 PU in 50 clock cycles.

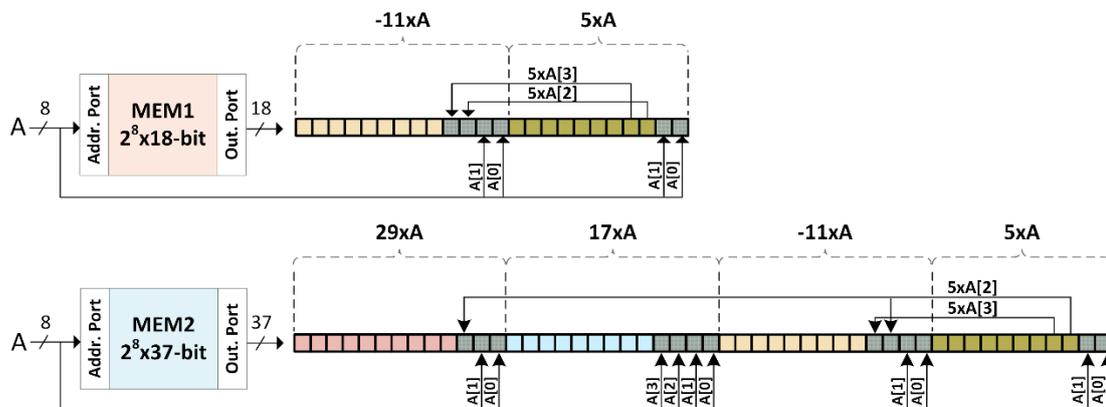


Figure 3.5 MEM1 and MEM2

3.3 Implementation Results

The proposed HEVC fractional interpolation hardware using memory based constant multiplication (FIHW_MEM) for all PU sizes is implemented using Verilog HDL. The Verilog RTL code is verified with RTL simulations. RTL simulation results matched the results of a software implementation of HEVC fractional interpolation algorithm.

The Verilog RTL code is synthesized and mapped to a Xilinx XC6VLX130T FF1156 FPGA with speed grade 3 using Xilinx ISE 14.7. FIHW_MEM FPGA implementation is verified to work at 233 MHz by post place and route simulations. Therefore, it can process 35 QFHD (3840x2160) video frames per second. It uses 3806 LUTs, 3815 DFFs and 1498 Slices.

In this thesis, three different HEVC fractional interpolation hardware implementations are used for energy consumption comparison. The first one (FIHW_ORG) is the original hardware proposed in [16]. It computes type A, B and C filters separately. The second one (FIHW_MCM) is the MCM hardware proposed in [17]. It computes multiplications with constant coefficients using Hcub MCM algorithm. The third one (FIHW_DSP) uses DSP blocks in FPGA for implementing multiplications with constant coefficients.

Verilog RTL codes of these three HEVC fractional interpolation hardware are synthesized and mapped to a Xilinx XC6VLX130T FF1156 FPGA with speed grade 3 using Xilinx ISE 14.7. FPGA implementation of FIHW_ORG uses 3752 LUTs, 3207 DFFs and 1848 Slices. FPGA implementation of FIHW_MCM uses 3370 LUTs, 3833 DFFs and 1543 Slices. FPGA implementation of FIHW_DSP uses 2747 LUTs, 3477 DFFs, 1406 Slices and 40 DSP48E1.

FPGA implementations of FIHW_ORG, FIHW_MCM and FIHW_DSP are verified to work at 154, 200 and 217 MHz, respectively, by post place and route simulations. Therefore, FPGA implementation of FIHW_ORG, FIHW_MCM and FIHW_DSP can process 23, 30 and 32 QFHD (3840x2160) video frames per second, respectively. The implementation results are shown in Table 3.2.

Table 3.2 Implementation Results

	FIHW_ORG	FIHW_MCM	FIHW_DSP	FIHW_MEM
FPGA	Xilinx Virtex6	Xilinx Virtex6	Xilinx Virtex6	Xilinx Virtex6
DFEs	3207	3833	3477	3815
LUTs	3752	3370	2747	3806
Slices	1848	1208	1406	1498
DSP48E1s	---	---	40	---
Max. Freq. (MHz)	154	200	217	233
Fps	23 QFHD	30 QFHD	32 QFHD	35 QFHD

Power consumptions of all FPGA implementations are estimated using Xilinx XPower Analyzer tool. Post place & route timing simulations are performed for Tennis, Kimono, Park Scene (1920x1080) video frames at 100 MHz [23] and signal activities are stored in VCD files. These VCD files are used for estimating power consumptions of the FPGA implementations. As shown in Figure 3.6, the proposed FIHW_MEM has up to 31%, 12.3% and 4.7% less energy consumption than FIHW_ORG, FIHW_MCM and FIHW_DSP, respectively.

Comparison of the proposed HEVC fractional interpolation hardware with the HEVC fractional interpolation hardware in the literature is shown in Table 3.3. The proposed HEVC fractional interpolation hardware has higher throughput than [16, 17, 26, 27]. Only hardware in [28] has higher throughput than the proposed hardware at the expense of more area. The hardware in [16] has less area than the proposed hardware. However, it can only be used for motion compensation.

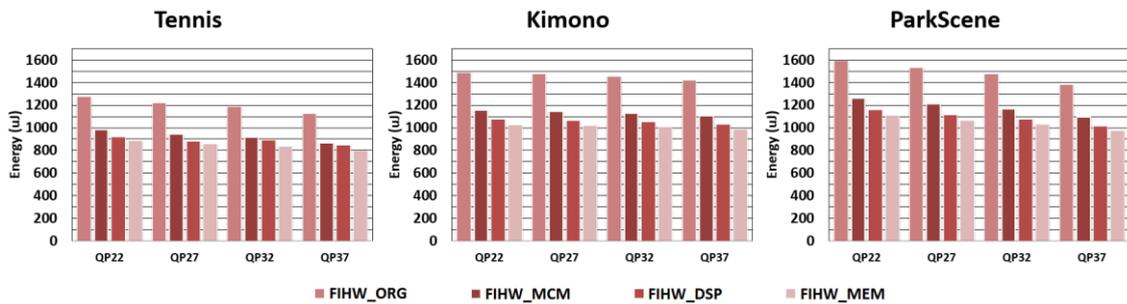


Figure 3.6 Energy Consumptions of FIHW_ORG, FIHW_MCM, FIHW_DSP and FIHW_MEM

Table 3.3 Hardware Comparison

	[16]	[17]	[26]	[27]	[28]	FIHW_MEM
FPGA	Xilinx Virtex 6	Xilinx Virtex 6	Arria II GX	Xilinx Virtex 5	Stratix III	Xilinx Virtex 6
Slices	---	---	---	2181	---	1498
LUTs	3005	3929	18831	5017	7701	3806
Block RAMs	2	6	---	2	---	---
Max. Freq. (MHz)	100	200	200	283	278	233
Fps	64	30	60	30	60	35
	2560x1600	3840x2160	1920x1080	2560x1600	3840x2160	3840x2160

CHAPTER IV

HIGH PERFORMANCE 2D TRANSFORM HARDWARE FOR FUTURE VIDEO CODING

HEVC uses DCT/IDCT. In addition, it uses DST/IDST for 4x4 intra prediction in certain cases. DCT and DST have high computational complexity, and they are heavily used in an HEVC encoder [10]. DCT and DST operations account for 11% of the computational complexity of an HEVC video encoder. They account for 25% of the computational complexity of an all intra HEVC video encoder [3, 29].

HEVC uses DCT-II and DST-VII. It uses 4x4, 8x8, 16x16, 32x32 TU sizes. In order to improve the compression efficiency, FVC uses DCT-II, DCT-V, DCT-VIII, DST-I, DST-VII, and it uses 4x4, 8x8, 16x16, 32x32, 64x64 TU sizes [11, 12]. Therefore, FVC transform operations have much higher computational complexity than HEVC transform operations.

In this thesis, three different high performance FVC 2D transform hardware are designed and implemented using Verilog HDL. They perform 2D DCT-II, DCT-V, DCT-VIII, DST-I, and DST-VII operations for 4x4 and 8x8 TU sizes by applying 1D transforms in vertical and horizontal directions. They process two 4x4 TUs in parallel or one 8x8 TU. Therefore, they can calculate 8 DCT/DST coefficients per clock cycle.

The first (baseline) hardware uses separate datapaths for each 1D transform. In this hardware, data gating is used to reduce energy consumption. In addition, Hcub MCM algorithm [18] is used to perform constant multiplications. Hcub MCM algorithm

reduces the number and size of adders. The second (reconfigurable) hardware uses one reconfigurable datapath for all 1D column transforms and one reconfigurable datapath for all 1D row transforms. Therefore, it has smaller area than the baseline hardware. However, the baseline hardware with data gating technique has less energy consumption than the reconfigurable hardware. This is because reconfigurable 1D datapath has larger area and more energy consumption than one baseline 1D datapath.

The third (reconfigurable_DSP) hardware uses one reconfigurable datapath for all 1D column transforms and one reconfigurable datapath for all 1D row transforms. 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 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. The reconfigurable_DSP hardware implements multiplications with constants using DSP blocks in FPGA instead of using adders and shifters. It uses data gating to reduce energy consumption.

Since it is more efficient to implement constant multiplications using adders and shifters instead of using multipliers in an ASIC implementation, the FVC baseline and reconfigurable hardware implement multiplications with constants using adders and shifters. Therefore, the FPGA implementation of reconfigurable_DSP hardware has up to 29% and 59% less energy consumption than FPGA implementations of baseline and reconfigurable hardware, respectively.

Several HEVC 2D DCT/IDCT hardware are proposed in the literature [29, 30, 31, 32, 33, 34]. The hardware proposed in [29, 30, 31, 32] implement HEVC DCT-II for TU sizes up to 32x32. In [30], DCT calculations are performed using multipliers. In [31], FPGA implementation of HEVC DCT-II is implemented using DSP blocks and ASIC implementation of HEVC DCT-II is implemented using multipliers. In [29] and [32], DCT calculations are done using adders and shifters. In [33], HEVC IDCT-II and IDST-VII are implemented using adders and shifters for TU sizes up to 32x32. In [34], FPGA implementation of HEVC DCT-II is proposed. This hardware uses DSP blocks for HEVC DCT-II operation. FVC 2D transform hardware proposed in this thesis are compared with the HEVC 2D DCT/IDCT hardware proposed in [29, 30, 31, 32, 33, 34]. Since FVC uses DCT-II, DCT-V, DCT-VIII, DST-I and DST-VII, FVC baseline,

reconfigurable and reconfigurable_DSP 2D transform hardware proposed in this thesis have larger area than the HEVC 2D transform hardware.

4.1 FVC Transform Algorithms

Basis functions for 1D DCT-II, DCT-V, DCT-VIII, DST-I and DST-VII for an $N \times N$ block are shown in Table 4.1, where $i, j = 0, 1, \dots, N-1$ [10].

Table 4.1 DCT-II, DCT-V, DCT-VIII, DST-I, DST-VII Basis Functions

Transform Type	Basis Function
DCT-II	$T_{ij} = \omega_0 \cdot \sqrt{\frac{2}{N}} \cdot \cos\left(\frac{\pi \cdot i \cdot (2j+1)}{2N}\right), \omega_0 = \begin{cases} \sqrt{\frac{2}{N}} & i = 0 \\ 1 & i \neq 0 \end{cases}$
DCT-V	$T_{ij} = \omega_0 \cdot \omega_1 \cdot \sqrt{\frac{2}{2N-1}} \cdot \cos\left(\frac{2\pi \cdot i \cdot j}{2N-1}\right), \omega_0 = \begin{cases} \sqrt{\frac{2}{N}} & i = 0 \\ 1 & i \neq 0 \end{cases}, \omega_1 = \begin{cases} \sqrt{\frac{2}{N}} & j = 0 \\ 1 & j \neq 0 \end{cases}$
DCT-VIII	$T_{ij} = \sqrt{\frac{4}{2N+1}} \cdot \cos\left(\frac{\pi \cdot (2i+1) \cdot (2j+1)}{4N+2}\right)$
DST-I	$T_{ij} = \sqrt{\frac{2}{N+1}} \cdot \sin\left(\frac{\pi \cdot (i+1) \cdot (j+1)}{N+1}\right)$
DST-VII	$T_{ij} = \sqrt{\frac{4}{2N+1}} \cdot \sin\left(\frac{\pi \cdot (2i+1) \cdot (j+1)}{2N+1}\right)$

HEVC uses DCT-II and DST-VII. It uses 4x4, 8x8, 16x16, 32x32 TU sizes for DCT. It also uses DST for 4x4 intra prediction in certain cases. HEVC performs 2D transform operation by applying 1D transforms in the vertical and horizontal directions. The coefficients in the HEVC 1D transform matrices are derived from the DCT-II and DST-VII basis functions. However, integer coefficients are used for simplicity. HEVC DCT-II and DST-VII matrices for 4x4 TU size are shown in (4.1) and (4.2).

In order to improve the compression efficiency, FVC uses DCT-II, DCT-V, DCT-VIII, DST-I, DST-VII, and it uses 4x4, 8x8, 16x16, 32x32, 64x64 TU sizes. FVC also performs 2D transform operation by applying 1D transforms in the vertical and horizontal directions. The coefficients in the FVC 1D transform matrices are derived

from DCT and DST basis functions. However, integer coefficients are used for simplicity. FVC transform matrices for 4x4 TU size are shown in (4.3)-(4.7).

$$DCT - II_{4x4} = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix} \quad (4.1)$$

$$DST - VII_{4x4} = \begin{bmatrix} 29 & 55 & 74 & 84 \\ 74 & 74 & 0 & -74 \\ 84 & -29 & -74 & 55 \\ 55 & -84 & 74 & -29 \end{bmatrix} \quad (4.2)$$

$$DCT - II_{4x4} = \begin{bmatrix} 256 & 256 & 256 & 256 \\ 334 & 139 & -139 & -334 \\ 256 & -256 & -256 & 256 \\ 139 & -334 & 334 & -139 \end{bmatrix} \quad (4.3)$$

$$DCT - V_{4x4} = \begin{bmatrix} 194 & 274 & 274 & 274 \\ 274 & 241 & -86 & -349 \\ 274 & -86 & -349 & 241 \\ 274 & -349 & 241 & -86 \end{bmatrix} \quad (4.4)$$

$$DCT - VIII_{4x4} = \begin{bmatrix} 336 & 296 & 219 & 117 \\ 296 & 0 & -296 & -296 \\ 219 & -296 & -117 & 336 \\ 117 & -296 & 336 & -219 \end{bmatrix} \quad (4.5)$$

$$DST - I_{4x4} = \begin{bmatrix} 190 & 308 & 308 & 190 \\ 308 & 190 & -190 & -308 \\ 308 & -190 & -190 & 308 \\ 190 & -308 & 308 & -190 \end{bmatrix} \quad (4.6)$$

$$DST - VII_{4x4} = \begin{bmatrix} 117 & 219 & 296 & 336 \\ 296 & 296 & 0 & -296 \\ 336 & -117 & -296 & 219 \\ 219 & -336 & 296 & -117 \end{bmatrix} \quad (4.7)$$

Table 4.2 shows the numbers of addition and shift operations required for calculating 1D DCT-II and DST-VII used in HEVC, and 1D DCT-II, DCT-V, DCT-VIII, DST-I and DST-VII used in FVC for 4x4 and 8x8 TU sizes. FVC transform operations have much higher computational complexity than HEVC transform operations.

Table 4.2 Addition and Shift Amounts

TU Size		Future Video Coding					HEVC	
		DCT-II	DCT-V	DCT-VIII	DST-I	DST-VII	DCT-II	DST-VII
4x4	Addition	88	248	224	160	224	64	200
	Shift	80	240	216	160	216	64	192
8x8	Addition	784	2232	2368	1008	2368	576	---
	Shift	608	2056	2176	912	2176	480	---

HEVC uses the same transform type for vertical and horizontal 1D transforms for performing a 2D transform. However, FVC may use different transform types for vertical and horizontal 1D transforms. It uses an AMT scheme to determine 1D transform types. AMT is enabled or disabled for each CU. When AMT is disabled for a CU, only DCT-II is used for this CU. When AMT is enabled for a CU, 1D transform types for vertical and horizontal directions are selected based on prediction type, intra or inter prediction, for this CU.

Table 4.3 Transform Sets

Transform Set	Transform Types
0	DST-VII, DCT-VIII
1	DST-VII, DST-I
2	DST-VII, DCT-V

In FVC, as shown in Table 4.3, three different 1D transform sets are defined [10]. Each transform set consists of two transform types. In intra prediction mode, transform set is selected based on intra prediction mode. In inter prediction mode, transform set 2 is used for all inter prediction modes.

4.2 Proposed FVC Baseline 2D Transform Hardware

The proposed FVC baseline 2D transform hardware for 4x4 and 8x8 TU sizes including Hcub MCM algorithm is shown in Figure 4.1. The proposed hardware performs 2D DCT/DST by first performing 1D DCT/DST on the columns of a TU, and then performing 1D DCT/DST on the rows of the TU. After 1D column DCT/DST, the resulting transformed coefficients are stored in a transpose memory, and they are used as input for 1D row DCT/DST. 1D column datapaths and 1D row datapaths are used to perform 1D column DCT/DST and 1D row DCT/DST operations, respectively.

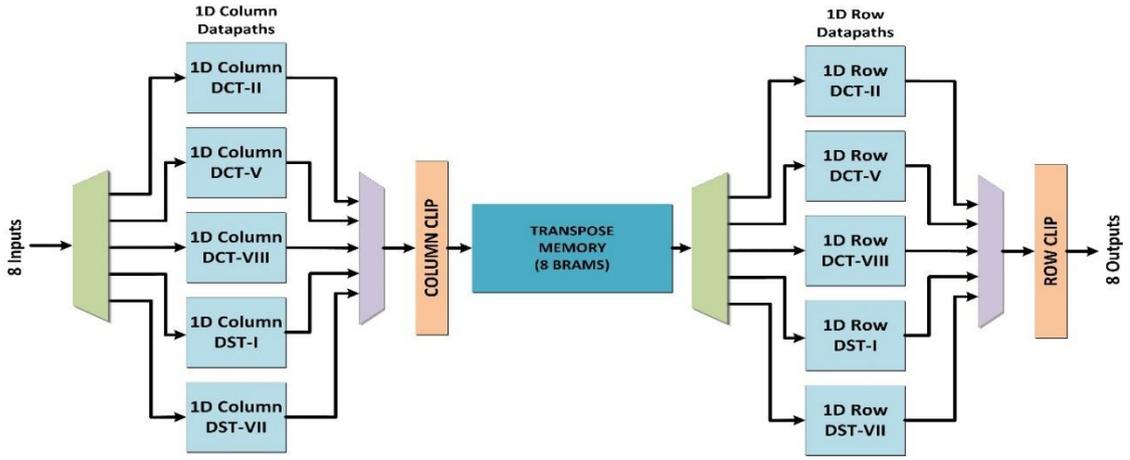


Figure 4.1 Proposed FVC Baseline 2D Transform Hardware

The proposed baseline hardware uses separate datapaths for implementing each 1D column and 1D row DCT/DST type. It processes two 4x4 TUs in parallel or one 8x8 TU. It calculates eight transformed coefficients per clock cycle for both 4x4 and 8x8 TU sizes. Proper inputs and outputs are selected based on the transform type selected for the current TU and its size. When the proposed hardware processes 8x8 TU size, eight inputs are eight residuals in one column of an 8x8 TU. When it processes 4x4 TU size, eight inputs are four residuals in one column of a 4x4 TU and four residuals in one column of another 4x4 TU.

An N-point 1D transform can be performed by performing two N/2-point 1D transforms with some preprocessing for FVC DCT-II and DST-I. FVC DCT-V, DCT-VIII and DST-VII do not have this property. In the proposed baseline hardware, N-point 1D DCT-II and 1D DST-I are performed by performing two N/2-point 1D DCT-II and 1D DST-I, respectively, with an efficient butterfly structure. N-point 1D DCT-V, 1D DCT-VIII and 1D DST-VII are performed by performing one N-point 1D DCT-V, 1D DCT-VIII and 1D DST-VII, respectively. The butterfly structure used for 1D DCT-II and 1D DST-I is shown in Figure 4.2. For 4x4 TUs, only 4x4 butterfly operation is used. For 8x8 TUs, 8x8 and 4x4 butterfly operations are used.

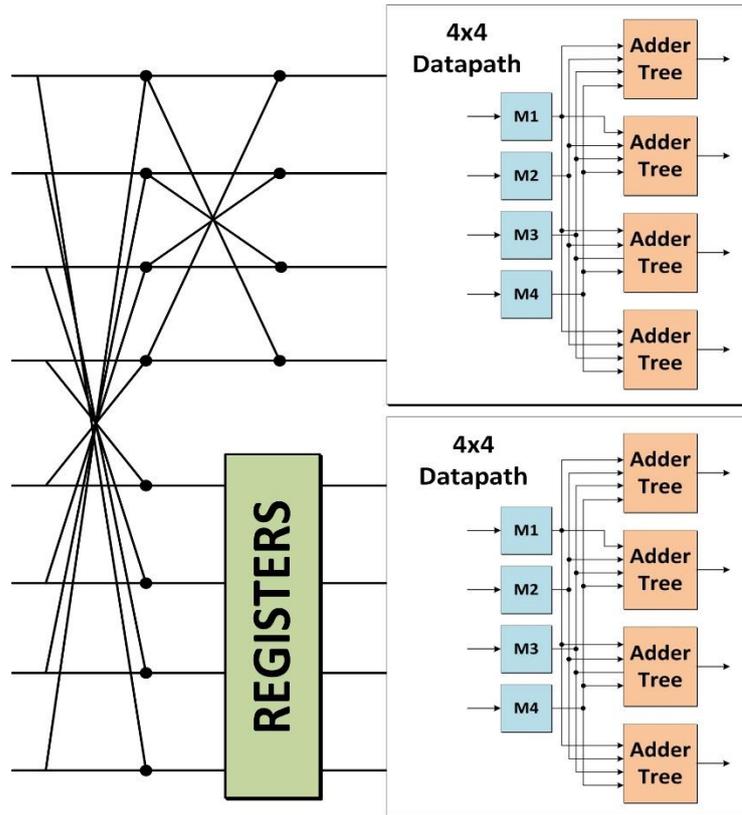


Figure 4.2 1D DCT-II/DST-I Column Datapath

In the proposed baseline hardware, there are eight 4x4 datapaths. As shown in Figure 4.2, there are two 4x4 datapaths in the 1D Column DCT-II, 1D Row DCT-II, 1D Column DST-I, 1D Row DST-I datapaths. Column and row datapaths have the same hardware architecture. Two 4x4 datapaths are used for two 4x4 TUs or for one 8x8 TU. In the proposed baseline hardware, there are six 8x8 datapaths. As shown in Figure 4.3, there is one 8x8 datapath in the 1D Column DCT-V, 1D Row DCT-V, 1D Column DCT-VIII, 1D Row DCT-VIII, 1D Column DST-VII, 1D Row DST-VII datapaths. There are 8 adder trees in an 8x8 datapath. In the figure, only one of them is shown for simplicity. Column and row datapaths have the same hardware architecture. One 8x8 datapath is used for two 4x4 TUs or for one 8x8 TU.

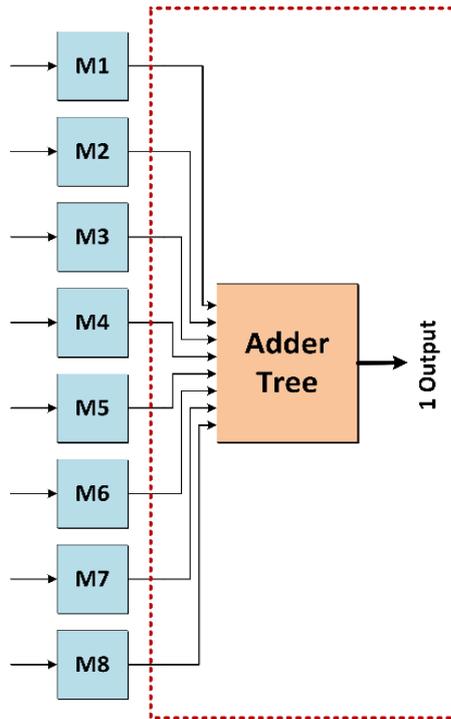


Figure 4.3 1D DCT-V/DCT-VIII/DST-VII Column Datapath

In order to reduce energy consumption of the proposed baseline hardware, data gating is used for the inputs of all 1D column datapaths and all 1D row datapaths. The input registers of the column and row datapaths for the transform types not selected for the current TU are not updated. This prevents unnecessary switching activities in these datapaths and therefore reduces energy consumption.

In the proposed baseline hardware, multiplications with constants are performed using adders and shifters. In order to reduce number and size of the adders, Hcub MCM algorithm is used [18]. Hcub MCM algorithm tries to minimize number and size of the adders in a multiplier block which multiplies a single input with multiple constants using addition and shift operations.

There are 4 multiplier blocks in a 4x4 datapath. Each multiplier block performs the multiplications between 1 input and 4 transform coefficients. One of the multiplier blocks in first 4x4 datapath for 1D Column DCT-II is shown in Figure 4.4. In order to calculate each output of 1D DCT-II and 1D DST-I for a 4x4 TU, an output from each multiplier block in a 4x4 datapath is selected, and these outputs are added or subtracted. In order to calculate each output of 1D DCT-II and 1D DST-I for an 8x8 TU, an output from each multiplier block in two 4x4 datapaths is selected, and these outputs are added or subtracted.

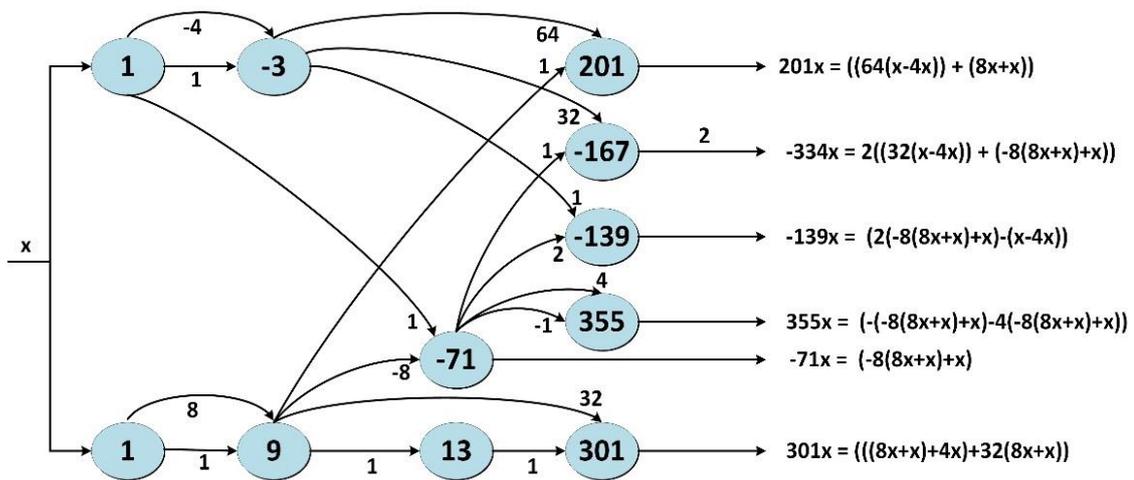


Figure 4.4 A Multiplier Block

There are 8 multiplier blocks in an 8x8 datapath. Each multiplier block performs the multiplications between 1 input and 8 transform coefficients. In order to calculate each output of 1D DCT-V, 1D DCT-VIII and 1D DST-VII for a 4x4 TU, an output from four multiplier blocks in an 8x8 datapath is selected, and these outputs are added or subtracted. In order to calculate each output of 1D DCT-V, 1D DCT-VIII and 1D DST-VII for an 8x8 TU, an output from each multiplier block in an 8x8 datapath is selected, and these outputs are added or subtracted.

As shown in Figure 4.5, the transpose memory is implemented using 8 Block RAMs (BRAM). 4 and 8 BRAMs are used for 4x4 and 8x8 TU sizes, respectively. Since a BRAM address can store 32-bits and one transformed coefficient of 1D column DCT/DST is 16-bits, each BRAM address can store two transformed coefficients. When the proposed hardware processes 4x4 and 8x8 TU size, each BRAM address stores two and one transformed coefficients, respectively.

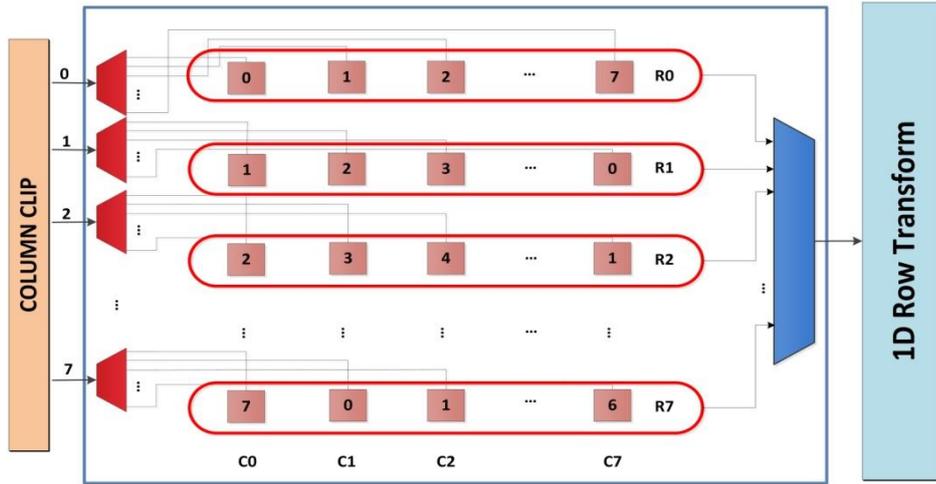


Figure 4.5 Transpose Memory

In the figure, the numbers in the each box show the BRAM that coefficient is stored. The results of 1D column DCT/DST are generated column by column. For 8x8 TU size, first, the coefficients in column 0 (C0) are generated in a clock cycle and stored in 8 different BRAMs. Then, the coefficients in column 1 (C1) are generated in the next clock cycle and stored in 8 different BRAMs using a rotating addressing scheme. This continues until the coefficients in column 7 (C7) are generated and stored in 8 different BRAMs using the rotating addressing scheme. This ensures that the 8 coefficients necessary for 1D row DCT/DST in a clock cycle can always be read in one clock cycle from 8 different BRAMs.

Column clip and row clip hardware are used to scale the outputs of 1D column DCT/DST and 1D row DCT/DST to 16 bits, respectively. Column clip hardware shifts 1D column DCT/DST outputs right by 3 and 4 bits for 4x4 and 8x8 TU sizes, respectively. Row clip hardware shifts 1D row DCT/DST outputs right by 10 and 11 bits for 4x4 and 8x8 TU sizes, respectively.

The proposed baseline hardware performs 1D DCT/DST for 4x4 and 8x8 TU sizes in 4 and 8 clock cycles, respectively. 1D column DCT/DST and 1D row DCT/DST operations are pipelined. While 1D row DCT/DST for current TU is performed, 1D column DCT/DST for next TU is also performed. Because of the input data loading and pipeline stages, the proposed baseline hardware starts generating the results of 1D row DCT/DST in 14 clock cycles. It then continues generating the results row by row in every clock cycle until the end of the last TU in the video frame without any stalls.

4.3 Proposed FVC Reconfigurable 2D Transform Hardware

The proposed FVC reconfigurable 2D transform hardware for 4x4 and 8x8 TU sizes is shown in Figure 4.6. Same as the proposed baseline hardware, it performs 2D DCT/DST by first performing 1D DCT/DST on the columns of a TU, and then performing 1D DCT/DST on the rows of the TU. The column clip hardware, row clip hardware, and transpose memory in the proposed reconfigurable hardware are the same as the ones in the proposed baseline hardware. Same as the proposed baseline hardware, it processes two 4x4 TUs in parallel or one 8x8 TU. It calculates eight transformed coefficients per clock cycle for both 4x4 and 8x8 TU sizes.

The proposed baseline hardware uses separate datapaths for implementing each 1D column and 1D row DCT/DST type. However, as shown in Figure 4.6, the proposed reconfigurable hardware uses one reconfigurable datapath for implementing all 1D column DCT/DST types and one reconfigurable datapath for implementing all 1D row DCT/DST types. Therefore, N-point DCT-II and DST-I are also performed by performing one N-point DCT-II and DST-I same as DCT-V, DCT-VIII, DST-VII.

Since, in FVC, one 1D DCT/DST at a time is performed, one reconfigurable datapath can be used for all 1D DCT/DST. 1D column datapath used in the proposed reconfigurable hardware is shown in Figure 4.7. Column and row datapaths have the same hardware architecture. There are 8 reconfigurable multiplier blocks in 1D column datapath. They perform the necessary constant multiplications for the selected 1D transform type (TR_Type_Veritical). In order to calculate each output of 1D DCT/DST for an 8x8 TU, an output from each reconfigurable multiplier block is selected, and these outputs are added or subtracted. There are 8 adder trees in the datapath. In the figure, only one of them is shown for simplicity.

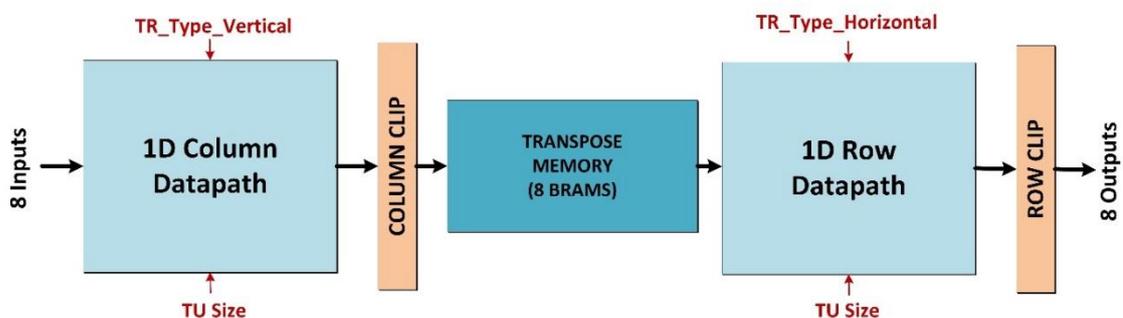


Figure 4.6 Proposed FVC Reconfigurable 2D Transform Hardware

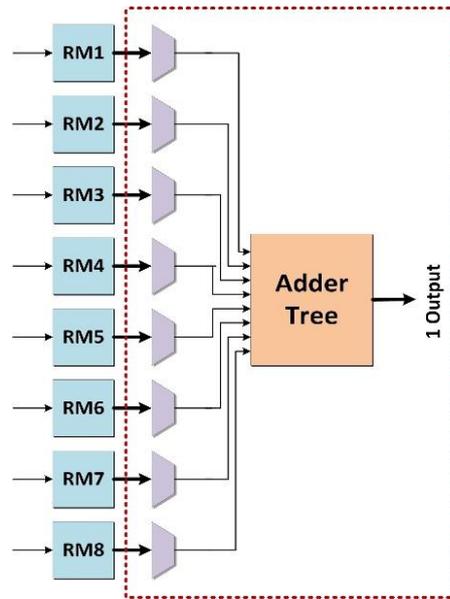


Figure 4.7 Reconfigurable 1D Column Datapath of the Proposed FVC Reconfigurable 2D Transform Hardware

The reconfigurable multiplier block is shown in Figure 4.8. Multiple constant multiplications necessary for calculating transformed coefficients for all 1D transform types and TU sizes have several common parts. First, multiplications with these common parts are performed in the common part of the reconfigurable multiplier block. Then, multiple constant multiplications necessary for calculating transformed coefficients for the selected 1D transform type and TU size are performed in the reconfigurable part of the reconfigurable multiplier block using the multiplication results of the common part.

The proposed reconfigurable hardware performs 1D DCT/DST for 4x4 and 8x8 TU sizes in 4 and 8 clock cycles, respectively. 1D column DCT/DST and 1D row DCT/DST operations are pipelined. While 1D row DCT/DST for current TU is performed, 1D column DCT/DST for next TU is also performed. Because of the input data loading and pipeline stages, the proposed reconfigurable hardware starts generating the results of 1D row DCT/DST in 14 clock cycles. It then continues generating the results row by row in every clock cycle until the end of the last TU in the video frame without any stalls.

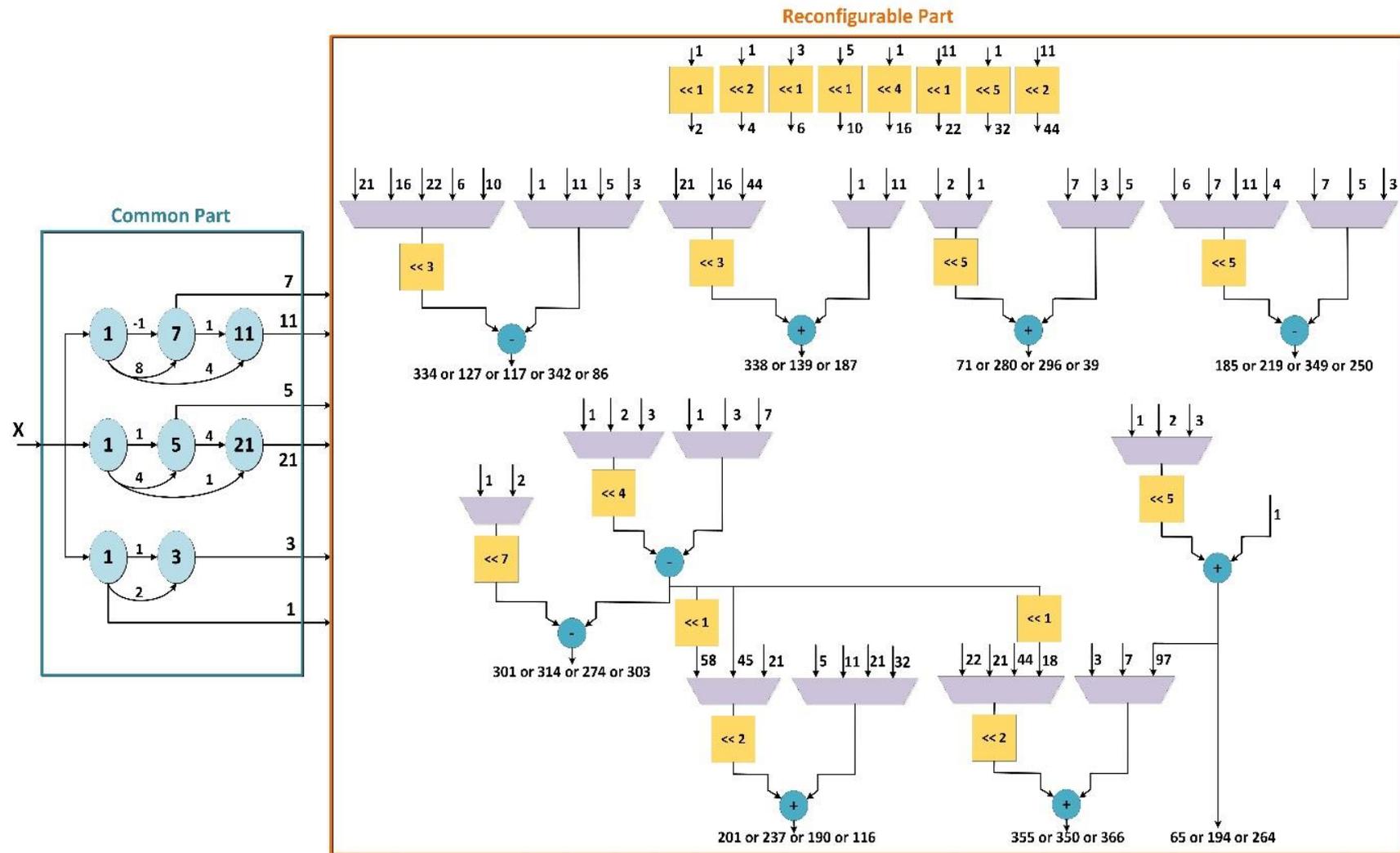


Figure 4.8 Reconfigurable Multiplier Block

4.4 Proposed FVC Reconfigurable_DSP 2D Transform Hardware

The proposed FVC reconfigurable_DSP 2D transform hardware for 4x4 and 8x8 TU sizes is shown in Figure 4.6. Same as the proposed baseline and reconfigurable hardware, it performs 2D DCT/DST by first performing 1D DCT/DST on the columns of a TU, and then performing 1D DCT/DST on the rows of the TU. The column clip hardware, row clip hardware, and transpose memory in the proposed reconfigurable_DSP hardware are the same as the ones in the proposed baseline and reconfigurable hardware. Same as the proposed baseline and reconfigurable hardware, it processes two 4x4 TUs in parallel or one 8x8 TU. It calculates eight transformed coefficients per clock cycle for both 4x4 and 8x8 TU sizes.

The proposed reconfigurable 1D column datapath used in the proposed reconfigurable_DSP hardware is shown in Figure 4.9. Column and row datapath have the same hardware architecture. Since each 1D DCT/DST uses different transform coefficients, different constant multiplication operations should be performed for each 1D DCT/DST. 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, the proposed hardware implements constant multiplications using DSP blocks in FPGA instead of using adders and shifters.

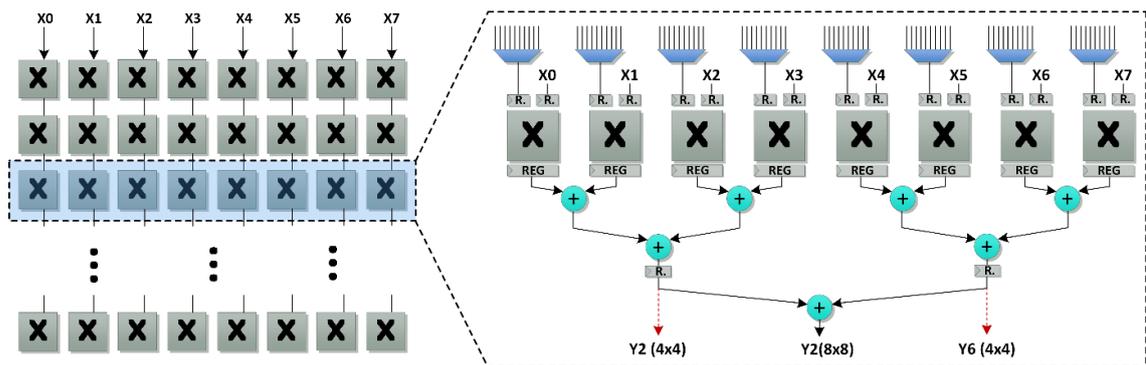


Figure 4.9 Reconfigurable 1D Column Datapath of the Proposed FVC Reconfigurable_DSP 2D Transform Hardware

For implementing constant multiplications, $8 \times 8 = 64$ DSP blocks are used in 1D column datapath and $8 \times 8 = 64$ DSP blocks are used in 1D row datapath. In the column datapath, each transform input sent to 8 DSP blocks in the same column. Each DSP block takes one transform input and one transform coefficient as input, and it performs constant multiplication. 64 and 32 DSP blocks are used for one 8×8 TU and two 4×4 TUs, respectively. Since the proposed hardware can perform 5 different DCT/DST operations for 2 different TU sizes, a multiplexer is used at the input of each DSP block to select proper transform coefficient. 1D transform type (TR_Type_Verical) and TU size (TU_size) are used as select signals for the multiplexers.

In order to calculate each output of 1D DCT/DST for an 8×8 TU, outputs of DSP blocks in the same row are added. 8 DSP blocks in the same row and their adder tree structure is shown in Figure 4.9. 8 DSP blocks in the other rows have the same structure. In the figure, only one of them is shown for simplicity.

In order to calculate each output of 1D DCT/DST for a 4×4 TU, outputs of DSP blocks in the same row are added. Since two 4×4 TUs are processed in parallel, outputs of first 4 DSP blocks in the same row are added for the first 4×4 TU. Outputs of last 4 DSP blocks in the same row are added for the second 4×4 TU.

In order to reduce energy consumption of the proposed hardware, data gating is used for the inputs of DSP blocks in 1D column datapath and 1D row datapath. 1D DCT/DST operation for an 8×8 TU uses 64 DSP blocks. 1D DCT/DST operation for a 4×4 TU uses 16 DSP blocks. Therefore, when two 4×4 TUs are processed in parallel, the input registers of 32 DSP blocks are not updated. This prevents unnecessary switching activities in the DSP blocks and therefore reduces energy consumption.

The proposed reconfigurable_DSP hardware performs 1D DCT/DST for 4×4 and 8×8 TU sizes in 4 and 8 clock cycles, respectively. 1D column DCT/DST and 1D row DCT/DST operations are pipelined. While 1D row DCT/DST for current TU is performed, 1D column DCT/DST for next TU is also performed. Because of input data loading and pipeline stages, the proposed hardware starts generating the results of 1D row DCT/DST in 16 clock cycles. It then continues generating the results row by row in every clock cycle until the end of the last TU in the video frame without any stalls.

4.5 Implementation Results

The proposed FVC baseline, FVC reconfigurable and FVC reconfigurable_DSP hardware are implemented using Verilog HDL. The Verilog RTL codes are verified

with RTL simulations. RTL simulation results matched the results of FVC 2D transform implementation in Joint Exploration Test Model (JEM) 4.0 reference software encoder [10]. The Verilog RTL codes are synthesized and mapped to a Xilinx XC6VLX550T FF1759 FPGA with speed grade 2 using Xilinx ISE 14.7. The FPGA implementations are verified with post place and route simulations. Post place and route simulation results matched results of FVC 2D transform implementation in JEM 4.0 reference software encoder.

An HEVC 2D DCT hardware for all TU sizes is proposed in [29]. In this thesis, two different versions of this hardware are implemented for 4x4 and 8x8 TU sizes, for fair comparison, using Verilog HDL. The first hardware (HEVC) uses Hcub MCM algorithm for multiplications with constants. The second hardware (HEVC_DSP) uses DSP blocks in FPGA for multiplications with constants.

Number of adders and shifters used in 1D (column or row) datapaths of FVC baseline, FVC reconfigurable and HEVC hardware are shown in Table 4.4. Hcub MCM algorithm considerably reduced number of adders and shifters in 1D datapaths of FVC baseline and HEVC hardware. The proposed FVC reconfigurable 1D column/row datapath uses significantly less adders and shifters than the proposed FVC baseline 1D column/row datapaths.

Number of multipliers, adders and multiplexers used in 1D (column or row) datapath of the proposed FVC reconfigurable_DSP hardware and the HEVC_DSP hardware are shown in Table 4.5. Since FVC 2D transform operations have much higher computational complexity than HEVC 2D DCT operations, reconfigurable 1D column/row datapath of the proposed FVC reconfigurable_DSP hardware uses more multipliers, adders and multiplexers than the column/row datapath in the HEVC_DSP hardware.

Table 4.4 Adder and Shifter Amounts in 1D Datapaths

			HEVC	FVC Baseline	FVC Reconfig.
Without MCM	Multiplier Blocks	Adder	60	817	112
		Shifter	80	996	248
	Adder Tree	Adder	28	224	56
With MCM	Multiplier Blocks	Adder	30	428	---
		Shifter	50	593	---
	Adder Tree	Adder	28	224	---

Table 4.5 Multiplier, Adder and Multiplexer Amounts in 1D Datapaths

	HEVC_DSP	FVC Reconfig._DSP Hardware
Multiplier	22	64
Adder	28	56
10-bit 2-to-1 MUX	---	342

The FPGA implementation results are shown in Table 4.6. Since the FPGA implementation of proposed FVC reconfigurable_DSP hardware uses DSP blocks for multiplications with constants, it uses less LUT and Slice than the FPGA implementation of proposed FVC baseline and reconfigurable hardware. Since FVC 2D transform operations have much higher computational complexity than HEVC 2D DCT operations, the proposed FVC reconfigurable_DSP hardware uses more resources than HEVC_DSP hardware.

The Verilog RTL codes of FVC baseline, FVC reconfigurable and HEVC hardware are also synthesized to a 90 nm standard cell library and the resulting netlists are placed and routed. Their gate counts are calculated according to NAND (3x1) gate area excluding on-chip memory. The ASIC implementation results are shown in Table 4.7.

Table 4.6 FPGA Implementation Results

	HEVC Hardware		HEVC_DSP Hardware	FVC Baseline Hardware		FVC Reconfig. Hardware	FVC Reconfig._DSP Hardware
	Without MCM	With MCM		Without MCM	With MCM		
FPGA	Xilinx Virtex6	Xilinx Virtex6	Xilinx Virtex6	Xilinx Virtex6	Xilinx Virtex6	Xilinx Virtex6	Xilinx Virtex6
Slices	1111	939	810	10215	7930	5292	1223
LUTs	3613	3119	2069	32586	27144	17173	3332
DFFs	1412	1065	665	15243	12309	4571	2082
Block RAMs	8	8	8	8	8	8	8
DSP48E1s	---	---	44	---	---	---	128
Max Freq. (MHz)	167	167	222	143	167	143	222
Fps	40 8K Ultra HD	40 8K Ultra HD	54 8K Ultra HD	35 8K Ultra HD	40 8K Ultra HD	35 8K Ultra HD	54 8K Ultra HD
TU Size	4, 8	4, 8	4, 8	4, 8	4, 8	4, 8	4, 8
Transform	2D	2D	2D	2D	2D	2D	2D

Table 4.7 ASIC Implementation Results

	HEVC Hardware		FVC Baseline Hardware		FVC Reconfig. Hardware
	Without MCM	With MCM	Without MCM	With MCM	
Technology	90 nm	90 nm	90 nm	90 nm	90 nm
Gate Count	29.3 K	28.5 K	153.4 K	136.4 K	60.1 K
Max Freq. (MHz)	200	200	333	345	245
Fps	48 8K Ultra HD	48 8K Ultra HD	82 8K Ultra HD	84 8K Ultra HD	60 8K Ultra HD
TU Size	4, 8	4, 8	4, 8	4, 8	4, 8
Transform	2D	2D	2D	2D	2D

Both ASIC and FPGA implementations of the proposed FVC reconfigurable hardware use less resources than the proposed FVC baseline hardware. Since FVC 2D transform operations have much higher computational complexity than HEVC 2D DCT operations, the proposed FVC baseline and reconfigurable hardware use more resources than HEVC hardware.

Power consumptions of the FPGA implementations are estimated using Xilinx XPower Analyzer tool. Post place & route timing simulations are performed for Tennis, Kimono and ParkScene full HD (1920x1080) videos at 100 MHz [35] and signal activities are stored in VCD files. These VCD files are used for estimating power consumptions of the FPGA implementations.

Energy consumptions of FVC baseline, FVC reconfigurable, FVC reconfigurable_DSP, HEVC and HEVC_DSP hardware for one frame of each video are shown in Figure 4.10. Data gating technique reduced the energy consumption of FVC baseline hardware up to 71.7%. Data gating technique and Hcub MCM algorithm together reduced the energy consumption of FVC baseline hardware up to 73.3%. Although the proposed FVC reconfigurable hardware has smaller area than the proposed FVC baseline hardware, it has more energy consumption than the proposed FVC baseline hardware when data gating technique is used. This is because reconfigurable 1D column/row datapath has larger area and more energy consumption than one baseline 1D column/row datapath. Since FVC 2D transform operations have much higher computational complexity than HEVC 2D DCT operations, the proposed FVC baseline and reconfigurable hardware consume more energy than HEVC hardware.

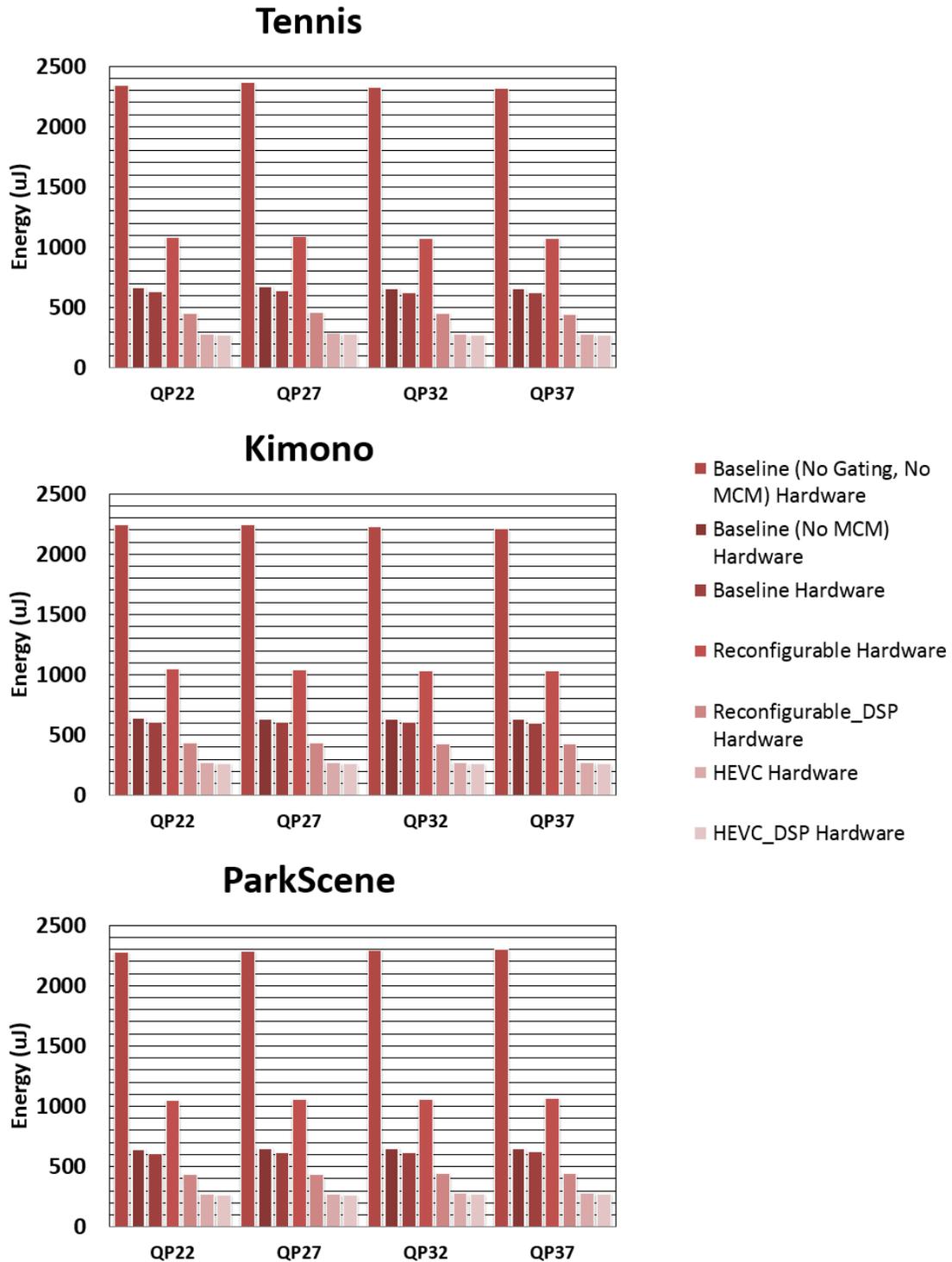


Figure 4.10 Energy Consumption Results

Since the proposed FVC reconfigurable_DSP hardware implements multiplications with constants using DSP blocks in FPGA instead of using adders and shifters, the proposed FVC reconfigurable_DSP hardware has up to 29% and 59% less energy consumption than the proposed FVC baseline and reconfigurable hardware,

respectively. Since FVC 2D transform operations have much higher computational complexity than HEVC 2D DCT operations, the proposed FVC reconfigurable_DSP hardware consumes more energy than HEVC_DSP hardware.

The comparison of FPGA implementations is shown in Table 4.8. Since FVC 2D transform operations have much higher computational complexity than HEVC 2D DCT operations, the FPGA implementations of FVC baseline, reconfigurable and reconfigurable_DSP hardware use more FPGA resources than the FPGA implementations of HEVC 2D DCT hardware proposed in [29, 31, 34]. Since HEVC 2D DCT hardware proposed in [31] performs DCT-II for TU sizes up to 32x32, its FPGA implementation uses more FPGA resources than the FPGA implementation of FVC reconfigurable_DSP hardware.

Table 4.8 Comparison of FPGA Implementations

	[31]	[34]	[29]		FVC Baseline	FVC Reconfig.	FVC Reconfig._DSP
			HEVC	HEVC_DSP			
FPGA	Arria II GX	Xilinx Virtex7	Xilinx Virtex 6	Xilinx Virtex 6	Xilinx Virtex 6	Xilinx Virtex 6	Xilinx Virtex 6
Slices	---	---	939	810	7930	5292	1223
LUTs	7300	2478	3119	2069	27144	17173	3332
DFFs	---	---	1065	665	12309	4571	2082
DSP48E1s	128	64	---	44	---	---	128
Max. Freq. (MHz)	200	289	167	222	167	143	222
Fps	---	70 3840x2160	40 7680x4320	54 7680x4320	40 7680x4320	35 3840x2160	54 7680x4320
Throughput (pixels/cycle)	---	---	8	8	8	8	8
Max Bit Length	25	25	25	25	27	27	27
TU Size	4, 8, 16, 32	4, 8	4, 8	4, 8	4, 8	4, 8	4, 8
Transform Type	DCT-II	DCT-II	DCT-II	DCT-II	DCT-II, DCT- V, DCT-VIII, DST-I, DST- VII	DCT-II, DCT- V, DCT-VIII, DST-I, DST- VII	DCT-II, DCT- V, DCT-VIII, DST-I, DST- VII
Transform	2D	2D	2D	2D	2D	2D	2D

The ASIC implementation results of the proposed FVC reconfigurable hardware are scaled up for all TU sizes in order to compare it with the HEVC 2D DCT/IDCT hardware in the literature. The comparison of ASIC implementations is shown in Table 4.9. Since FVC 2D transform operations have much higher computational complexity than HEVC 2D transform operations, the proposed FVC reconfigurable hardware has larger area than HEVC hardware in the literature. HEVC 2D transform hardware proposed in [32] has higher performance than the proposed FVC reconfigurable

hardware. However, although it only performs HEVC DCT-II transform, it has similar area with the proposed FVC reconfigurable hardware. The proposed FVC reconfigurable hardware has higher performance than the other HEVC 2D transform hardware.

Table 4.9 Comparison of ASIC Implementations

	[29]	[30]	[31]	[32]	[33]	FVC Reconfigurable Hardware
Technology	90 nm	90 nm				
Gate Count	175 K	343.5 K	328.2 K	347 K	142 K	416 K
Max Freq. (MHz)	140	311	400	187	150	160
Fps	60	30	30	60	48	39
	3840x2160	4096x2048	3840x2160	7680x4320	3840x2160	7680x4320
Throughput (pixels/cycle)	4/8/16/32	4/8/16/32	8/16/32/32	32	4/8/16/32	---
Max Bit Length	25	25	25	25	25	27
TU Size	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32 (Scaled)
Transform Type	DCT-II	DCT-II	DCT-II	DCT-II	IDCT-II, IDST-VII	DCT-II, DCT-V, DCT-VIII, DST-I, DST-VII
Transform	2D	2D	2D	2D	2D	2D

4.6 Implementation on FPGA Board

In this thesis, the proposed FVC reconfigurable 2D transform hardware is implemented on a ML605 FPGA board which includes a Virtex 6 XC6VLX240T FPGA with speed grade 1, 512 MB DDR RAM and 32 MB Flash memory and interfaces such as UART.

Xilinx Platform Studio, Xilinx Software Development Kit and MicroBlaze processor are used for implementing the proposed FVC reconfigurable 2D transform hardware on the FPGA board. A software running on MicroBlaze processor is developed to send the inputs of the proposed FVC reconfigurable 2D transform hardware from a host to the hardware and to read the outputs of the hardware for sending them back to the host computer. The proposed FVC reconfigurable 2D transform hardware is added as a peripheral to a bus where the MicroBlaze processor is the master. For this purpose, the proposed FVC reconfigurable 2D transform hardware is modified to be a slave peripheral for this data bus and software accessible registers are added to the proposed hardware. These registers are used by the software running on

MicroBlaze for writing data inputs and communication signals to the hardware. They are also used for reading the outputs and the status information from the hardware.

The software reads residual data of one frame from the host computer using the UART interface and writes it to a DDR RAM. Then, it loads residual data for one TU to the input registers of the hardware with TU size information. The proposed FVC reconfigurable 2D transform hardware produces transform coefficients and writes them to the output registers. After the hardware sends done signal to the software, the software reads transform coefficients from the output registers and writes them to the DDR RAM. This process is repeated for all TUs in one frame. Since the produced transform coefficients are not pixel values, they are not displayed on monitor. The produced transform coefficients are read from DDR RAM and it is verified that they matched the results of FVC 2D transform implementation in JEM 4.0 reference software encoder. The FPGA implementation is shown in Figure 4.11.

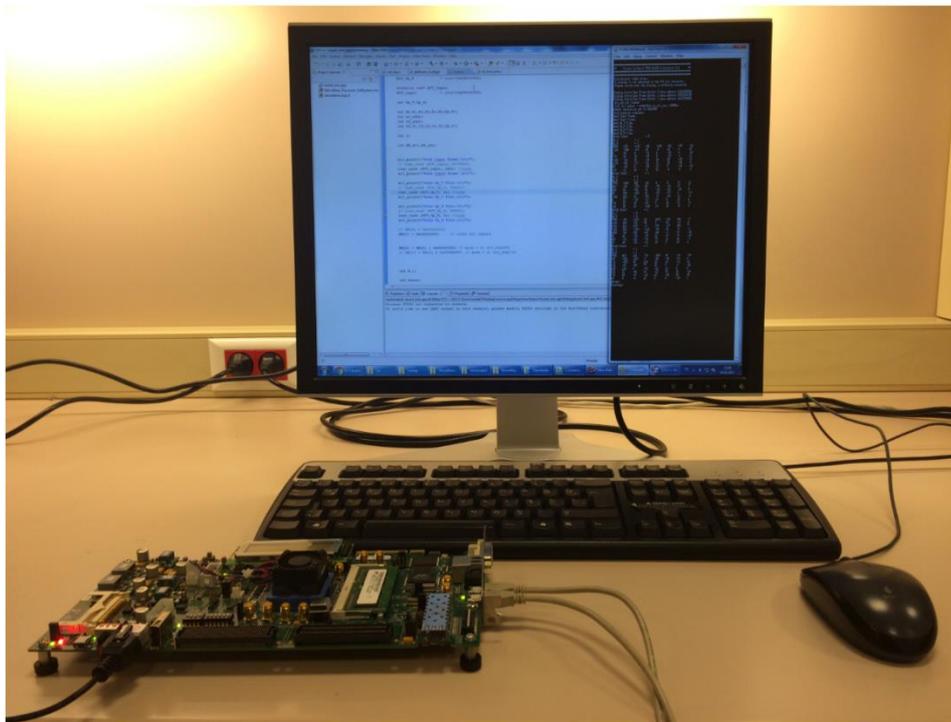
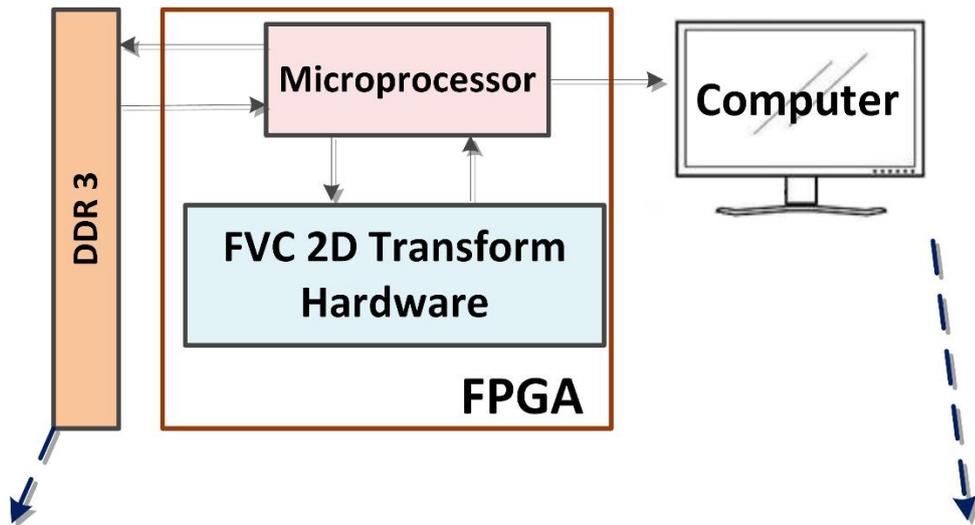


Figure 4.11 Proposed FVC Reconfigurable 2D Transform Hardware Implementation on FPGA Board

CHAPTER V

CONCLUSIONS AND FUTURE WORK

In this thesis, we proposed a low complexity HEVC SPME technique for SPME in HEVC encoder. The proposed technique reduced the amount of computations significantly with slight decrease in PSNR. We designed and implemented a high performance HEVC SPME hardware implementing the proposed low complexity HEVC SPME technique. We also designed and implemented an HEVC fractional interpolation hardware using memory based constant multiplication for all PU sizes for both HEVC encoder and decoder. The proposed hardware uses memory based constant multiplication technique for implementing multiplications with constant coefficients. We proposed three different high performance FVC 2D transform hardware for 4x4 and 8x8 TU sizes. The first two hardware use adders and shifters for implementing FVC transform algorithm. The third hardware uses DSP blocks in Xilinx Virtex 6 FPGA for implementing FVC transform algorithm. The proposed hardware is verified to work correctly on an FPGA board.

As future work, rate-distortion performance of the proposed low complexity HEVC SPME technique can be determined using HM reference software encoder. Memory based constant multiplication hardware used in the proposed HEVC fractional interpolation hardware can be implemented more efficiently to further reduce energy consumption. The proposed FVC 2D transform hardware can be extended to implement all TU sizes, 16x16, 32x32 and 64x64.

BIBLIOGRAPHY

- [1] B. Bross, W.J. Han, J.R. Ohm, G.J. Sullivan, Y.K. Wang, and T. Wiegand, "High Efficiency Video Coding (HEVC) Text Specification Draft 10", *JCTVC-L1003*, Feb. 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] ITU-T and ISO/IEC, High Efficiency Video Coding, *ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC)*, April 2003.
- [4] F. Bossen, B. Bross, K. Suhring and D. Flynn, "HEVC Complexity and Implementation Analysis ", *IEEE Trans. on Circuits and Systems for Video Technology*, vol.22, no.12, pp.1685-1696, Dec. 2012.
- [5] J. Vanne, M. Viitanen, T.D. Hämäläinen 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.
- [6] C.K. Huang, L.C. Wu, H.T. Huang, T.H. Sheng, L.L. Youn, "A Low-Power High-Performance H.264/AVC Intra-Frame Encoder for 1080p HD Video", *IEEE Trans. on Very Large Scale Integration Systems*, vol.19, no.6, pp.925-938, June 2011.
- [7] G. J. Sullivan, G. Bjøntegaard, and A. Luthra T. Wiegand, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.
- [8] 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.

- [9] J. Chen, Y. Chen, M. Karczewicz, X. Li, H. Lu, L. Zhang, X. Zhao, “Coding tools investigation for next generation video coding”, *ITU-T SG16 COM16-C806*, Feb. 2015.
- [10] J. Chen, E. Alshina, G. J. Sullivan, J. R. Ohm, J. Boyce, “Algorithm Description of Joint Exploration Model 4”, *JVET-D1001*, Oct. 2016.
- [11] X. Zhao, J. Chen, M. Karczewicz, L. Zhang, X. Li, W. Chien, “Enhanced Multiple Transform for Video Coding”, *Proc. Data Compression Conference*, April 2016.
- [12] T. Biatek, V. Lorcy, P. Castel, P. Philippe, “Low-Complexity Adaptive Multiple Transform for Video Coding”, *Proc. Data Compression Conference*, April 2016.
- [13] A. C. Mert, E. Kalali, I. Hamzaoglu, “Low complexity HEVC sub-pixel motion estimation technique and its hardware implementation”, *IEEE Int. Conference on Consumer Electronics – Berlin*, Sept. 2016.
- [14] A. C. Mert, E. Kalali, I. Hamzaoglu, “High Performance 2D Transform Hardware for Future Video Coding”, *IEEE Trans. on Consumer Electronics*, vol. 62, no. 2, May 2017.
- [15] A. C. Mert, E. Kalali, I. Hamzaoglu, “An FPGA Implementation of Future Video Coding 2D Transform”, *IEEE Int. Conference on Consumer Electronics – Berlin*, Sept. 2017.
- [16] E. Kalali, Y. Adibelli, I. Hamzaoglu, “A Reconfigurable HEVC Sub-Pixel Interpolation Hardware”, *IEEE Int. Conference on Consumer Electronics - Berlin*, Sept. 2013.
- [17] E. Kalali, I. Hamzaoglu, “A low energy HEVC sub-pixel interpolation hardware,” *IEEE Int. Conference on Image Processing*, pp. 1218-1222, Oct. 2014.
- [18] Y. Voronenko, M. Püschel, “Multiplierless Constant Multiple Multiplication”, *ACM Trans. on Algorithms*, vol. 3, no. 2, May 2007.
- [19] V. Afonso, H. Maich, L. Audibert, B. Zatt, M. Porto, L. Agostini, “Memory-Aware and High-Throughput Hardware Design for the HEVC Fractional Motion Estimation”, *Symposium on Integrated Circuits and System Design*, 2013.
- [20] G. He, D. Zhou, Y. Li, Z. Chen, T. Zhang, S. Goto, “High-Throughput Power-Efficient VLSI Architecture of Fractional Motion Estimation for Ultra-HD HEVC Video Encoding”, *IEEE Trans. on VLSI Systems*, vol.23, no.12, pp.3138-3142, March 2015.
- [21] D. Ding, X. Ye, S. Wang, “1/2 and 1/4 Pixel Paralleled FME with A Scalable Search Pattern for HEVC Ultra-HD Encoding”, *IEEE Int. Conf. on Communication Technology*, pp.278-281, Oct. 2015.
- [22] K. McCann, B. Bross, W.J. Han, I.K. Kim, K. Sugimoto, G. J. Sullivan, “High Efficiency Video Coding (HEVC) Test Model (HM) 15 Encoder Description”, *JCTVC-Q1002*, June 2014.

- [23] F. Bossen, "Common test conditions and software reference configurations", *JCTVC-11100*, May 2012.
- [24] P. K. Meher, "LUT Optimization for Memory-Based Computation", *IEEE Transactions on Circuits and Systems-II: Express Briefs*, vol. 57, no. 4, Apr. 2010.
- [25] P. K. Meher, "New Approach to Look-Up-Table Design and Memory-Based Realization of FIR Digital Filter", *IEEE Transactions on Circuits and Systems-I: Regular Papers*, vol. 57, no. 3, Mar. 2010.
- [26] G. Pastuszak, M. Trochimiuk, "Architecture Design and Efficiency Evaluation for the High-Throughput Interpolation in the HEVC Encoder", *16th Euromicro Conference on Digital System Design*, Sept. 2013.
- [27] C. M. Diniz, M. Shafique, S. Bampi, 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.
- [28] H. Maich, C. Afonso, D. Franco, B. Zatt, M. Porto, L. Agostini, "High throughput hardware design for the HEVC Fractional Motion Estimation Interpolation Unit", *IEEE 20th International Conference on Electronics, Circuits, and Systems*, May 2014.
- [29] E. Kalali, A. C. Mert, 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.
- [30] J. Zhu, Z. Liu, D. Wang, "Fully Pipelined DCT/IDCT/Hadamard Unified Transform Architecture for HEVC Codecs", in *Proc. IEEE Int. Symp. on Circuits and Systems*, May 2013.
- [31] G. Pastuszak, "Hardware Architecture for the H.265/HEVC Discrete Cosine Transform", *IET Image Processing*, vol. 9, no. 6, pp. 468-477, June 2015.
- [32] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim, C. Yeo, "Efficient Integer DCT Architectures for HEVC", *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 24, no. 1, pp. 168-178, Jan. 2014.
- [33] E. Kalali, E. Ozcan, O. M. Yalcinkaya, I. Hamzaoglu, "A Low Energy HEVC Inverse Transform Hardware", *IEEE Trans. on Consumer Electronics*, vol. 60, no. 4, pp. 754-761, Nov. 2014.
- [34] M. Chen, Y. Zhang, C. Lu, "Efficient architecture of variable size HEVC 2D-DCT for FPGA platforms", *AEU-International Journal of Electronics and Communications*, vol. 73, pp. 1-8, March 2017.

[35] K. Suehring, X. Li, “JVET Common Test Conditions and Software Reference Configurations”, *JVET-B1010*, Feb. 2016.