Energy-Efficient Architecture for the Affine Transforms

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—Modern video encoders such as the Versatile Video Coder (VVC) are capable of superior compression compared with their predecessors, enabling high-quality video transmission even on bandwidth-limited networks. However, this gain comes at the cost of expensive algorithms, including flexible partitioning and advanced temporal prediction techniques. VVC brings a new motion compensation model based on affine transformations that are capable of representing complex movements like rotation and zoom. This extends the traditional translation-based model and is highly beneficial for compression efficiency, but it also brings even higher complexity requirements due to its complex operations, which include pixel interpolation. This paper proposes hardware architectures for the 4-parameter and the 6-parameter affine transforms of VVC encoders. The proposed architectures were designed to provide high throughput while keeping an efficient consumption. Synthesis results targeting a 45 nm? standard cell technology are shown and compared between implementations.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

According to a study by Cisco, more IP traffic is expected in the next five years than in the entire history of the Internet. By 2022, 60% of the world's population will be Internet users. More than 28 billion devices and connections will be online. And video will account for 82% of all IP traffic [1].

Video communications will require continuous improvement in compression efficiency and image quality as data consumption rapidly increases. Video consumption has increased dramatically in recent years, requiring increasingly efficient algorithms and implementations to enable the use of increasingly complex video compressors.

Recognizing the growing demand for more efficient compression, a new video coding standardization project called versatile video encoding (VVC) was released in 2018. VVC provides 30% to 50% better compression than the HEVC standard while maintaining the same quality of experience [2] and supports new compression features such as 4K to 16K resolutions and a new motion compensation model based on affine transforms.

Video codecs such as HEVC use only the translational motion model. However, in the real world, the motion of an object as seen by the camera can combine multiple types of motion. Versatile Video Coding (VVC) uses affine motion prediction and compensation with 4 and 6 parameters, and AV1, which uses warped motion compensation, so more complex motion can be computed in these newer video codecs.

The affine transformation is a geometric transformation that preserves lines and parallelism and allows rotation, scaling, translation, and shear. These capabilities increase the complexity of compression. High compression complexity is associated with high processing overhead. VVC simplifies this model by using pixel blocks instead of individual pixels, resulting in lower memory usage and complexity. Each macroblock consists of many 4x4 pixel sub-blocks. Each sub-block has its own macro-block-derived motion vector estimate (MV), which in VVC can range from a single sub-block (4x4 pixels) to a 128x128 block (1024 sub-blocks) [3]. In addition to this technique, the latest encoders have much higher complexity.

The new AV1 encoder achieves the best compression results among its competitors (including the current industry standard HEVC), but at the cost of 2500 to 3000 times slower compared to the others [4]. These results show that real-time coding is a major challenge for AV1 encoders. Therefore, solutions that provide computational optimizations for this pattern are extremely important. Although VVC has higher efficiency than previous models, nowadays real-time compression is required due to the demand for live streams such as movies, TV, and YouTube videos. In video streaming, content is sent over the Internet in compressed form and displayed by the viewer in real time. Hardware solutions that improve video compression efficiency are suitable for such fast compression, but real-time video compression to compensate for network delays is a difficult implementation challenge even with fast processor architectures [5].

This paper is organized as follows. In Section II, we give a brief introduction to the mathematical operation of affine transformation and its implications. Section III shows our proposed architectures for performing affine transforms in real-time. Experimental results are discussed in Section IV, and Section V summarizes and concludes this paper.

II. Affine Motion Compensation

This section is divided in 3 parts. In part A the affine operation will be introduced and it's implications. Secondly, part B we'll get deeper in mathematical operations for each

motion transformation, and finally, the last part will discuss the operations involved to compute all the affine transform.

A. Affine Motion Compensation

An affine transformation is a transformation in which collinearity is preserved, so that all points that originally meet on a line remain on a line after the transformation, and in which the distance relationships are preserved (the midpoint of a line segment remains the average point even after the transformation).

Geometric contraction, expansion, dilation, reflection, rotation, shear, similarity transformations, spiral similarities, and translation are all affine transformations, as are their combinations. In general, an affine transformation is a composite of rotations, translations, dilations, and shear.

This freedom of transformations, combined with the simplicity of computations, has led to the affine transformation attracting more attention compared to other high-order motion models [6].

In an affine transformation, the proportions on the lines are preserved, but not necessarily preserve the angles or lengths. Any triangle can be transformed into any other by an affine transformation so that all triangles are affine, and in this sense affine is a generalization of congruent and similar.

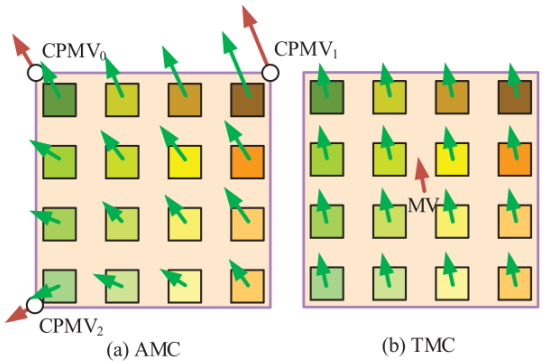

Fig. 1. legend (Source: [7])

Unlike classical translational motion compensation (TMC), affine motion compensation (AMC) uses multiple control motion vectors to derive the motion vector. This greater complexity allows AMC characterize more efficiently than non-translational motions TMC [7]. Affine motion estimation (AME) enables the prediction of not only translational motion, but also linearly transformed motion such as scaling and rotation. When a camera zooms or rotates while recording a video, AMC can predict the motion more accurately than translation-based [8].

B. Mathematical Functions

Prior to the Affine transformation step, the Affine prediction step takes place. In the prediction, the selection of candidate input vectors that can actually represent some of the movements covered by the transformation takes place. After this selection of vectors, they will serve as input to the system, necessary for the transformation calculations [6], according to equations 1, 2, 3, 4 and Figure 2.

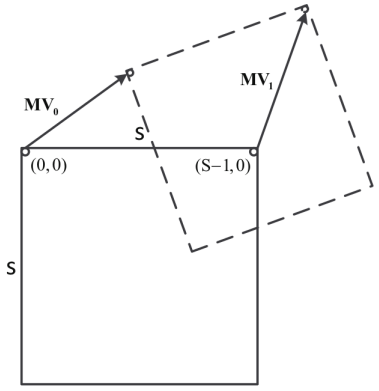

Fig. 2. Affine Motion Vectors representation (Source: [7])

$MV^h_{(x,y)}$ and $MV^v_{(x,y)}$ constitute the horizontal and vertical vectors of the Affine transformation and define the position, size and estimated rotation of the block used. MV_0 and MV_1 constitute the input vectors of the system and are the vectors selected by the prediction step produced in the Motion Estimation step of neighboring blocks in a 4-parameter AME. An example of selected vectors (for a 4-parameter affine motion model) can be seen in Figure 2.

Before the operation is performed, one of the affine transformation modes is selected based on the chosen vectors MV_0 and MV_1 and MV_2 (or $CPMV_0$ and $CPMV_1$ and $CPMV_2$ as shown in Figure 1). The two affine transformation modes existing modes are:

- 4-parameter affine: with only 2 vectors it can perform transformations by translation, rotation and scaling.
- 6-parameter affine: with 3 vectors (as in Figure **??**) you can do everything the 4-parameter transformation can do, but with the additional of shearing and aspect ratio transformations.

$$MV^h_{(x,y)} = \frac{MV^h_1 - MV^h_0}{W-1}x + \frac{MV^v_1 - MV^v_0}{W-1}y + MV^h_0 \quad (1)$$

$$MV^v_{(x,y)} = \frac{MV^v_1 - MV^v_0}{W-1}x + \frac{MV^h_1 - MV^h_0}{W-1}y + MV^v_0 \quad (2)$$

$$MV^h_{(x,y)} = \frac{MV^h_1 - MV^h_0}{W-1}x + \frac{MV^h_2 - MV^h_0}{H-1}y + MV^h_0 \quad (3)$$

$$MV^v_{(x,y)} = \frac{MV^v_1 - MV^v_0}{W-1}x + \frac{MV^v_2 - MV^v_0}{H-1}y + MV^v_0 \quad (4)$$

Eq. 1 and 2 represents the 4-parameter affine motion model, while Eq. 3 and 4 represents the 6-parameter affine motion model.

C. Involved Operations

The Affine transformation consists of 2 main steps:

1) Motion Vector Generation

2) Sub-pixel Interpolation

As shown in Figure 1(a), based on the input vectors ($CPMV_0$ and $CPMV_1$ and $CPMV_2$), each pixel subblock (2x2 pixels in VVC) has its motion subvector represented by the green arrows. These subvectors indicates where each subblock will be placed, and consequently, each pixel locations. These subvectors are used by the interpolation step to create pixels that are missing after the transformation due to the nature of the transformation, which can result in regions without defined pixels.

III. Proposed Architecture

To facilitate the visualization and understanding of the proposed architecture, we can abbreviate the terms of the equations 1, 2, 3, 4 by:

$$a = \frac{MV_1^h - MV_0^h}{W} \tag{5}$$

$$b = \frac{MV_1^v - MV_0^v}{W} \tag{6}$$

$$d = \frac{MV_2^h - MV_0^h}{H} \tag{7}$$

$$e = \frac{MV_2^v - MV_0^v}{H} \tag{8}$$

$$c = MV_0^h, \quad f = MV_0^v \tag{9}$$

A. Motion Vector Generator (MVG)

In the figure below 3, it's possible to see the motion vector generator architecture, this architecture generates an equivalent vector for each x,y and horizontal and vertical motion vector.

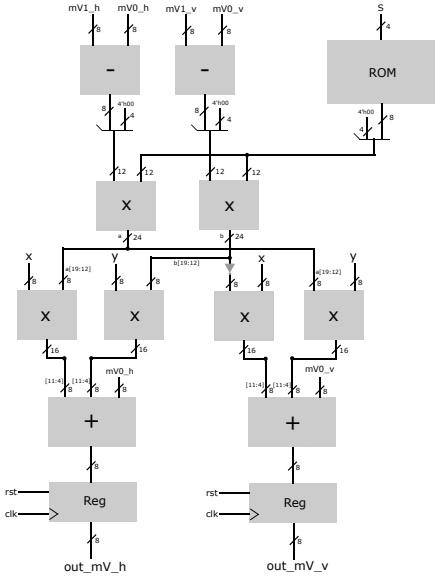

Fig. 3. Motion Vector Generator (MVG) architecture

This architecture implements the Eq. 1 and 2, the $1/(W-1)$ operation is implemented by a multiplication, because it can be synthesized. The results are early computed and stored in the ROM module.

The output of this block is in series with the interpolation block.

B. Interpolation

The proposed architecture for sub-pixel interpolation is shown in Fig. 4 where the blue lines represent the 16 bits width wires. An integer pixel value from the block reference buffer is given as an input to this combinational architecture. The motion vectors for horizontal (mv_h) and vertical (mv_v) interpolation are passed by the MVG block and are represented in an 8-bit fixed point thus the 4 most significant bits are integers and the other less significant bits are fractionals which select the corresponding filter. The blocks from A0 to A14 are the transposed coefficients for the 8-tap filter which interpolates the integer pixels by multiplying and adding them.

To efficiently perform the multiplications within the A's blocks, a multiplerless multiplication was implemented using shifts, additions and subtractions. It could be done because the filter coefficients are constants therefore in this manner the energy consumption is reduced compared to general purpose multipliers [9].

IV. Results and Discussion

This blocks was synthesized separately using TSMC 45nm and the results are shown in the table (1) below:

TABLE I
SYNTHESIS RESULTS

Block	Power	Area	Critical Path
Motion Vctor Generator	40 uW	1589	4.29 ns
Interpolator	2.4 mW	4696.65	2.58 ns

V. Conclusion

References

[1] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper," 2019.

[2] B. Bross, K. Andersson, M. Bläser, V. Drugeon, S.-H. Kim, J. Lainema, J. Li, S. Liu, J.-R. Ohm, G. J. Sullivan *et al.*, "General video coding technology in responses to the joint call for proposals on video compression with capability beyond hevc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 5, pp. 1226–1240, 2019.

[3] Y.-W. Huang, J. An, H. Huang, X. Li, S.-T. Hsiang, K. Zhang, H. Gao, J. Ma, and O. Chubach, "Block partitioning structure in the vvc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3818–3833, 2021.

[4] J. . J. . Ozer, "Wikipedia - MSU Codec Comparison 2017," 2018. [Online]. Available: https://en.wikipedia.org/wiki/AV1

[5] M. Tahir, "Fast video encoding using spatio-temporal features and ensemble classification," Ph.D. dissertation, CAPITAL UNIVERSITY, 2019.

[6] L. Li, H. Li, D. Liu, Z. Li, H. Yang, S. Lin, H. Chen, and F. Wu, "An efficient four-parameter affine motion model for video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1934–1948, 2017.

[7] D. Jin, J. Lei, B. Peng, W. Li, N. Ling, and Q. Huang, "Deep affine motion compensation network for inter prediction in vvc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 6, pp. 3923–3933, 2021.

[8] S.-H. Park and J.-W. Kang, "Fast affine motion estimation for versatile video coding (vvc) encoding," *IEEE Access*, vol. 7, pp. 158 075–158 084, 2019.

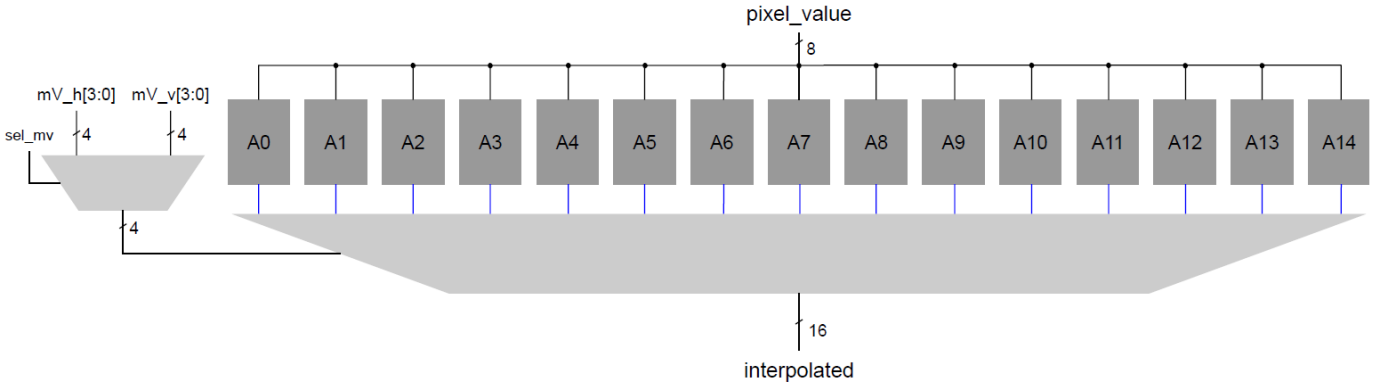

Fig. 4. Diagram of the proposed sub-pixel interpolation architecture.

[9] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007.