Chapter 1

Joint algorithm-architecture design of customizable modules for future video coding

Cláudio M. Diniz[1], Brunno Abreu[2], Mateus Grellert[3], Felipe Martin Sampaio[4], Daniel Palomino[5], Fábio Luís Livi Ramos[6], Bruno Zatt[7] and Sergio Bampi[8]

1.1 Introduction

Along the last three decades, video coding algorithms have been evolving driven by the popularization of digital videos and by the non-stopping demand for higher quality, larger resolutions, larger frame rates, and immersion. Starting from a niche market in early 1990s, digital videos became omnipresent with applications ranging from digital cinema to personal recording including video surveillance, telemedicine, video conferencing, video streaming, etc. Video coding experts have been enabling such evolution through the release of generations of video coding standards. H.261, published in 1988, and MPEG-1, published in 1993, represent the first video coding standards of a long-standing series that include MPEG-2, H.263, MPEG-4 Part 2, H.264/AVC, and the state-of-the-art HEVC, released in 2013.

The evolution of video coding standards brought impressive coding efficiency improvements. However, as one can expect, such efficiency comes at the cost of increased computational complexity. For instance, the encoding complexity increase observed from H.264/AVC to its successor, the HEVC, ranges from 1.2× to 3.2× [1]. In turn, the current implementation of the upcoming Versatile Video Coding (VVC) standard is up to 18.8× more complex than the reference implementation of HEVC (as shown in Section 1.3). Therefore, there is a need for high-throughput video coding solutions capable of employing optimized/simplified algorithms with no/reduced coding efficiency losses.

[1]Catholic University of Pelotas
[2]Federal University of Rio Grande do Sul
[3]Catholic University of Pelotas
[4]Federal Institute of Rio Grande do Sul
[5]Federal University of Pelotas
[6]Federal University of Pampa
[7]Federal University of Pelotas
[8]Federal University of Rio Grande do Sul

According to CISCO [2], 71% of the worldwide internet video traffic will be generated/consumed using mobile devices by 2022. Given the battery/energy restrictions of such mobile devices, energy-efficient video (de)coding becomes mandatory leading to the need for dedicated hardware architectures. However, some algorithms employed by coding standards are not suitable for hardware implementation demanding hardware-friendly optimizations. As a result, it has been demonstrated [3, 4, 5] that high-throughput energy-efficient video coding systems can be achieved by jointly designing algorithms and hardware architectures.

Although every new video coding generation brings new tools, features, and particularities, they follow a common hybrid video coding template. In this sense, it is possible to learn from challenges related to current video coding generation in order to anticipate the characteristics of future video coding standards and design efficient custom architectures.

This chapter presents a variety of custom hardware modules conceived though joint algorithm-architecture design for the HEVC standard. Based on that and considering experiments performed over VVC software, we map such solutions to the context of future video coding and highlight the major challenges to be faced in the near future. A discussion regarding the video coding evolution and its state of the art, including a qualitative comparison between HEVC and VVC, is presented in Section 1.2. Section 1.3 brings an experimental comparison between HEVC and VVC reference software implementations highlighting coding efficiency and computational complexity perspectives. In order to bring a comprehensive discussion, this chapters covers most of the major modules of the video encoder: rate-distortion optimization (Section 1.4), inter-frame prediction (Section 1.5), intra-frame prediction (Section 1.6), transforms (Section 1.7), in-loop filter (Section 1.8), and entropy coding (Section 1.9). Conclusions and upcoming challenges are presented in Section 1.10.

1.2 Video coding evolution and state of the art

1.2.1 Evolution of video coding standards

The most popular international video coding standards were developed along three decades by International Telecommunication Union (ITU-T) Video Coding Experts Group (VCEG) and by International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) Motion Picture Experts Group (MPEG). H.261 [6] is the first video coding standard developed by the VCEG. Other popular standards developed by ISO/IEC and/or ITU-T are MPEG-1 [7], H.262/MPEG-2 [8], H.263 [9], MPEG-4 Part 2 [10], and H.264/AVC (Advanced Video Coding) [11].

The most efficient video coding standard is High Efficiency Video Coding (HEVC) [12] (ITU-T H.265 and ISO/IEC MPEG-H part 2). HEVC was developed by the Joint Collaborative Team on Video Coding (JCT-VC), formed by experts of VCEG and MPEG, to support ultra-high resolution video (beyond 1920x1080 pixels) with better compression than H.264/AVC. HEVC standard achieves 40% bit rate reduction (on average) compared to H.264/AVC for a similar objective video quality [1]. The support of larger block sizes (to deal with increased spatial resolution), the in-

troduction of new flexible quadtree block partitioning structure and several advanced coding tools contribute to achieve such result [13]. However, HEVC encoder software (HEVC Test Model - HM) [14] is up to 3.2X more complex than H.264/AVC's Joint Test Model (JM) encoder [1].

Other video codecs exist, e.g. VP8, VP9, Daala, and Thor. Recently, Alliance for Open Media (AOM) is developing the royalty-free AV1 codec [15]. Recent analysis with 28 video sequences shows that AV1 encoder is 6.8X more complex than HM encoder to achieve a similar compression efficiency [16]. Another work found similar result and concludes that AV1 cannot transform its increased complexity into compression efficiency [17]. This chapter will focus on future standardization efforts of ITU-T and ISO/IEC.

The increasing demand for increased resolution and different video formats, such as high dynamic range and 360-degree videos, led ISO/IEC and ITU-T to establish the Joint Video Exploration Team (JVET) in October 2015 to study a new standard to succeed HEVC. After evidences collected from experiments on the Joint Exploration Model (JEM), JVET issued in October 2017 a Call for Proposals [18] with the goal to reduce bit rate of approximately 50% at the same subjective quality compared to HEVC [19]. Twenty-two groups responded to the call for proposals by April 2018. After that, JVET was renamed to Joint Video Experts Team, and named the new standard as Versatile Video Coding (VVC). The first VVC draft [20] and reference software called VVC Test Model 1 (VTM 1) [21, 22] started with a minimal set of common coding tools from the received proposals, while other tools are still under test. The publication of the standard is planned to 2020. The next section overviews and compares HEVC and VVC codecs.

1.2.2 Overview of HEVC and VVC codecs

Upcoming VVC video codec keeps the same hybrid video compression concept that is used in previous video coding standards, e.g. H.264/AVC and HEVC, a block-based codec with prediction, transforms and entropy coding. Fig. 1.1 shows the main components of a video encoder that follows this concept. First, the captured input video frames, represented in YCbCr color format, are partitioned into blocks of square size. Each block is processed by the encoder modules shown in Fig. 1.1.

Prediction, Transforms, Quantization, and Entropy coding compose the forward path of video encoder. Intra-frame prediction generates a predicted block of samples based on reconstructed neighbor samples of the current frame. Inter-frame prediction is formed by Motion Estimation (ME) and Motion Compensation (MC), that generate a prediction based on samples from reference frames. ME searches in the reference frames for a block that is the most similar with the current block to be encoded. A motion vector, that indicates the displacement between the most similar block (best block) and current block positions is sent in the bitstream. MC reconstructs the block samples indicated by the motion vector. A residue, i.e. the differences between predicted block and original block, is calculated and delivered to Transforms and Quantization module. Transforms convert the samples to frequency domain to decorrelate and concentrate the energy in a few coefficients. Quantization eliminates some small coefficients that are not relevant to the human eye, incurring

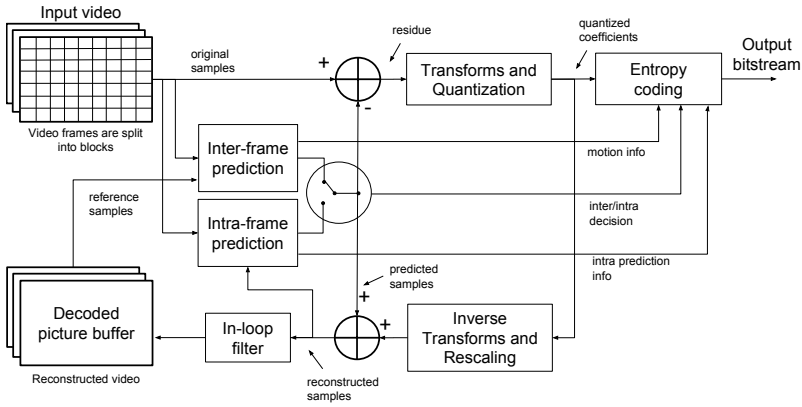


Figure 1.1: Generic video encoder diagram.

into coding losses. The quantization is controlled by a Quantization Parameter (QP), that is directly proportional to the strength of coding loss, and controls the quality of reconstructed video. Every coding information (motion vectors, intra prediction modes, intra/inter mode decision and residue) is delivered to Entropy coding module that generates the final bitstream of coded video, which a certain bitrate.

The inverse path, which avoids mismatch of the reconstructed frames between encoder and decoder, is formed by Inverse Transforms and Rescaling followed by In-loop Filter. Inverse Transforms convert quantized coefficients back to spatial domain that are further re-scaled to the magnitude level of predicted samples. Intra-frame prediction is applied to unfiltered reconstructed samples. Inter-frame prediction uses input from In-Loop Filter that is applied to remove blocking artifacts caused by a strong quantization step. The output samples of In-loop filter module are stored into a Decoded Picture Buffer (DPB), which are used as reference frames by inter-frame prediction. Since reconstructed frames are different from original frames (because of quantization), the quality loss must be evaluated with quality metrics, such as the Peak Signal-to-Noise Ratio (PSNR).

Fig. 1.2 shows the main components of a video decoder. It looks like the inverse path of video encoder. The coded video bitstream input is delivered to Entropy decoding module. The quantized coefficients are processed by Inverse Transforms and Rescaling module, while other information, e.g. motion vectors, intra-frame prediction modes, are delivered the prediction modules. Inter-frame prediction contains only the MC module. Predicted samples are added to the output of Inverse Transforms and Rescaling to reconstruct blocks that are filtered by In-loop filter to produce the decoded video to be displayed and stored into DPB to be used as reference frames for inter-frame prediction.

The main differences between HEVC and VVC standard is observed within the main codec modules, as shown in Table 1.1. It is important to note that that HEVC standard is concluded, so the coding tools are defined in [12] and does not change. On the other hand, VVC is under development, thus some coding tools may

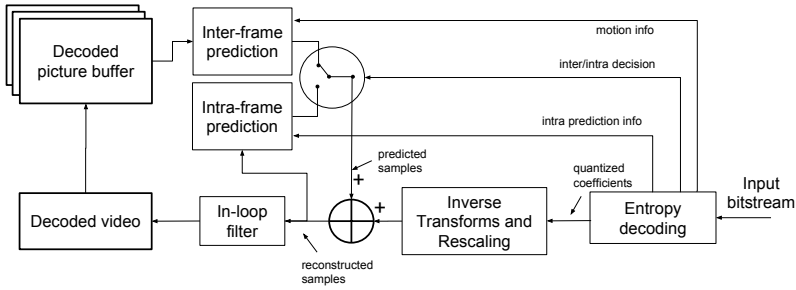


Figure 1.2: Generic video decoder diagram.

be included or removed until the conclusion of the standard. Hence, we consider the coding tools included in the VVC Draft 1 and in VTM 1 [20, 22].

Table 1.1 Tools currently supported by the HEVC and VVC reference encoders

	HEVC (HM) [12]	VVC (VTM) [20, 22]
Block sizes (luma samples)	64×64 - 4×4	128×128 - 4×4
Partitioning Structures	CU, PU, TU in a CTU quadtree structure	CTU quadtree with nested multi-type binary/ternary tree
Intra-frame Prediction	33 directional modes, DC and planar modes	65 directional modes, DC and planar modes
Inter-frame Prediction	1/4-pixel luma MV precision, 1/8-pixel chroma MV precision, merge mode	affine motion compensation, up to 1/16 MV precision (in affine mode), adaptive motion vector resolution (AMVR)
Transforms	32×32 - 4×4 DCT, 4×4 DST	64×64 - 2×2 DCT, DST
Quantization	QP = [0, 51]	QP = [0, 63]
Entropy	CABAC	CABAC
In-loop filter	SAO, Deblocking filter	SAO, Deblocking filter, ALF

HEVC and VVC divide video frames into Coding Tree Units (CTUs), composed by one Luma Coding Tree Block (CTB) and two Chroma CTBs. The maximum size of Luma CTB in HEVC is 64×64 samples. Chroma CTBs has the maximum size of 32×32 samples when 4:2:0 color subsampling format is used. VVC increased the maximum CTB sizes to 128×128 luma samples and 64×64 chroma samples to support higher spatial resolutions.

In HEVC, CTUs can be recursively partitioned in a quadtree manner into one or four Coding Units (CU) with a restriction that the minimum size of Luma CTB is 8×8 samples. Leaf CUs must be further partitioned into Prediction Units (PU) and Transform Units (TU) (as independent quadtrees) with the partitioning modes shown in Fig. 1.3 with a minimum size of luma CTBs of 4×4 samples. Some partitioning modes are restricted to Inter PUs. Transform sizes define the size of TUs ranging from 32×32 down to 4×4 luma samples.

In VVC, a CTU is partitioned into CUs using a quadtree with nested multi-type tree structure, i.e. the quadtree leaf CUs nodes are partitioned into binary or ternary

partitions, as shown in Fig. 1.3. There are no PUs and TUs in VVC, so the leaf CUs are used for prediction and transforms without any further partitioning, unless leaf CU is larger than the maximum transform size of 64×64 samples. The minimum size of luma CTB is 4×4 samples. VVC increased the number of partitioning ways because the binary and ternary partitions are not restricted to Inter PUs, but can be used to intra prediction and transforms (restricted to the maximum transform size is 64×64 luma samples and the minimum block size is of 4×4 luma samples).

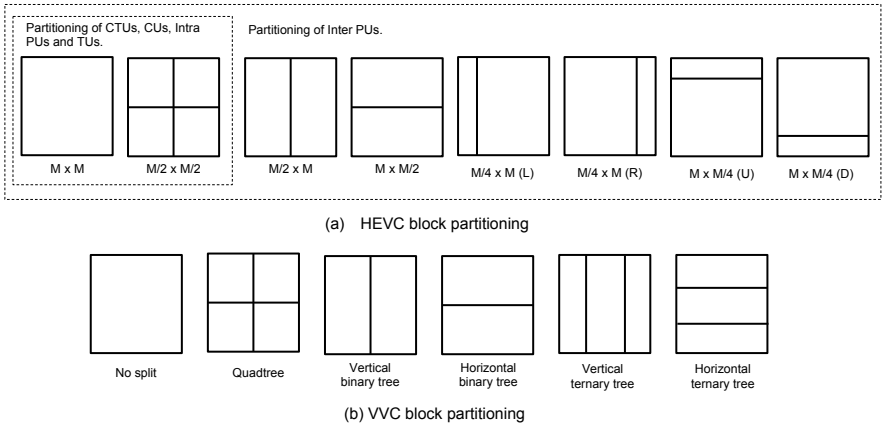


Figure 1.3: Block partitioning in (a) HEVC and (b) VVC standards.

Other coding tools included in HEVC and VVC are detailed in the following sections. Before them, we conduct an application analysis and comparison between HEVC and VVC reference softwares in Section 1.3.

1.3 Video coding application analysis

As previously stated, the VVC encoder aims at achieving a 50% compression gain for the same image quality compared with its predecessor, the HEVC. In order to achieve this, new tools are constantly being tested and compared with the current implementation in terms of coding complexity and compression gain. The tools presented in Table 1.1 are the ones currently implemented in the reference software implementation of each encoder, namely HM [14] and VTM [21].

If we compare both implementations based solely on the supported tools, it is natural to expect a higher complexity in VVC encoders, as more partitioning possibilities, intra directions, inter-prediction modes and transforms are supported. To mitigate this issue, the VTM software supports parallel processing as well as Single Instruction/Multiple Data (SIMD) optimizations. The SIMD kernels include interpolation filters, buffer allocation and distortion metrics, and none of them has any impact in the coding efficiency. The parallelism tools, however, break dependencies in the CABAC context tables, which affects the encoding performance to some extent.

This section will discuss how the HM and VTM encoders compare in terms of encoding complexity and compression efficiency. Table 1.2 presents the setup used to obtain the results presented in this discussion. The QP values and GOP structure are the ones recommended in the Common Test Conditions (CTC) document published by JCT-VC [23].

Table 1.2 Setup for experiments

HEVC encoder	HM 16.9
VVC encoder	VTM 1.1 with SIMD optimizations
QP	22, 27, 32, 37
Frame count	150
Temporal structure	Random Access
Computing Platform	Intel Core i5-7400 CPU @ 3.00 GHz, 4 GB RAM

The Bjontegaard Delta bitrate (BD-rate) metric will be used to compare both encoders in terms of compression efficiency [24]. The BD-rate value compares a reference and a test encoder, measuring the average bitrate difference between them for the same image quality. This metric is computed using the following equation:

$$BD\text{-}rate(E_{ref}, E_{test}) = \frac{\int_a^b BR_{ref} - BR_{test}}{b-a} \tag{1.1}$$

In (1.1), BR_{ref} and BR_{test} are the bitrate values obtained at each point of the integration interval (a,b). The rate-distortion curves are obtained using a piece-wise cubic Hermite interpolating polynomial with at least four (bitrate, PSNR) pairs, one for each QP value (22, 27, 32, 37).

1.3.1 Analysis of VVC and HEVC encoders

1.3.1.1 Compression efficiency

The first set of results presented in this section show how the VVC standard compares with its previous version in terms of compression. The results presented in Table 1.3 represent the BD-rate for the Y and YUV PSNR values, as well as the ratio between the VTM and the HM encoding time. In this analysis, the SIMD optimizations were enabled, as the main goal was to assess the compression efficiency of both encoders.

It is possible to observe that the VVC encoder is still far from its 50% improvement goal, achieving an average BD-rate gain of 13.1% (Y) and 14.8% (YUV). One of the reasons for the higher gain in the YUV BD-rate, which is a weighted combination of luminance and chrominance PSNRs, is likely the fact that chrominance CTBs may have a different tree structure than luminance ones in intra-predicted frames [22].

Another observation from Table 1.3 is that the results do not seem to be greatly affected by spatial resolution, but scene characteristics are still playing a major factor in the efficiency of the new standard. This goes against the versatile characteristic that is being targeted by the JVET group, showing that novel solutions capable of achieving a similar coding performance under diverse input sequences are necessary.

Table 1.3 BD-rate and encoding time ratio between the HM and the VTM encoders

Resolution	Sequence	BD-rate Y	BD-rate YUV	Time ratio[1]
2560x1600	PeopleOnStreet	-12.76%	-14.13%	2.54
	Traffic	-18.5%	-19.95%	1.17
	SteamLocomotive	-14.82%	-18.41%	1.70
	NebutaFestival	-4.72%	-10.59%	2.27
1920x1080	BasketballDrive	-16.91%	-19.02%	2.39
	ParkScene	-14.23%	-15.33%	1.37
	Kimono	-11.87%	-12.71%	2.28
	Cactus	-13.65%	-15.2%	1.85
	BQTerrace	-11.35%	-14.39%	1.19
832x480	BQMall	-14.28%	-15.45%	1.91
	RaceHorsesC	-10.23%	-11.4%	2.51
	BasketballDrill	-16.89%	-18.17%	2.06
	PartyScene	-12.98%	-13.91%	1.99
416x240	BasketballPass	-13.56%	-14.66%	1.94
	BQSquare	-11.44%	-12.44%	1.19
	BlowingBubbles	-12.83%	-13.93%	1.66
	RaceHorses	-11.43%	-12.29%	2.35
	Average	-13.09%	-14.82%	1.91

[1] VTM using SIMD optimizations

Using SIMD optimizations, the VTM encoder takes 1.91× more time to encode sequences on average. Of course this is not a fair comparison, but it already shows that the compression-complexity trade-off remains a problem in this new encoder. The next section will elaborate on this analysis using an unoptimized VTM encoder.

1.3.1.2 Computational effort

This section will assess how encoding computational effort[9] scales from encoder to another across the different encoding steps. The valgrind tool set [25] was used to dynamically instrument the HM and VTM references, accounting the cycles spent in each function. The VTM parallelism tools were not accounted for in this analysis, as this would compromise the confidence of our measurements. The SIMD optimizations were also disabled.

Fig. 1.4 shows a bar chart containing the main kernels and the estimated cycles spent using the HM and VTM encoders. The white-filled boxes indicate the complexity increase factor of each tool. To obtain these results, 30 frames of each sequence were encoded using the Random Access structure and a QP of 22. The reduced number of frames was necessary to obtain results in feasible time, because valgrind increases processing time by a factor of 4 on average due to its dynamic instrumentation.

Observing Fig. 1.4 we can conclude that the VTM encoder is more complex than HM in practically every encoding step. Overall, VTM took 3.4 and 5.4 more time to encode the BQSquare and the RaceHorses sequences respectively.

[9]In this chapter, computational effort and complexity will be used interchangeably, both meaning processing time of the encoder applications.

The increase factors with values above the average were written in bold in Fig. 1.4. It is possible to observe that the FME interpolation, the IME search, and the intra-prediction search had a significant increase in complexity in this new standard.

The encoding time difference between the BQSquare (left part of Fig. 1.4) and the RaceHorses (right part) sequences is a product of several coding tools, but the chart evidences that most of it lies in the Integer Motion Estimation (IME) search, which includes the fast Test Zone (TZ) search and the bi-prediction refinement. For the BQSquare sequence, the encoding complexity of this step increased by 4.28× compared with HM, whereas the increase was 18.77× when encoding RaceHorses. One of the reasons for such disparity comes from the fast search algorithm implemented in both HM and VTM, which takes longer to converge in scenes with complex motion.

The intra-prediction search should also be pointed out, even though this is not visually noticeable in the chart. By comparing HM and VTM, intra prediction search takes 8.09× and 11.05× more cycles to be computed in the latter for the BQSquare and RaceHorses sequences respectively. Though this does not seem to pose any harm in this particular example, as the intra-prediction is responsible for less than 3% of the overall encoding time, some applications require all frames to be intra-coded (e.g., surveillance footage), so the VTM intra-prediction is clearly an important optimization target as well.

The reference implementations [14, 21] used in this analysis serve as basis to assess the coding efficiency of the supported methods, but are not necessarily followed in practical applications, since standards only define the decoding process. The following sections will discuss how video-coding researchers exploit this fact to design efficient hardware and software solutions to mitigate encoding processing and energy requirements while keeping coding efficiency close to its reference.

1.4 Rate-Distortion Optimization

Rate-Distortion Optimization (RDO) is applied at the encoder to decide the coding modes that minimize the rate and distortion of coded video [26]. RDO is formulated

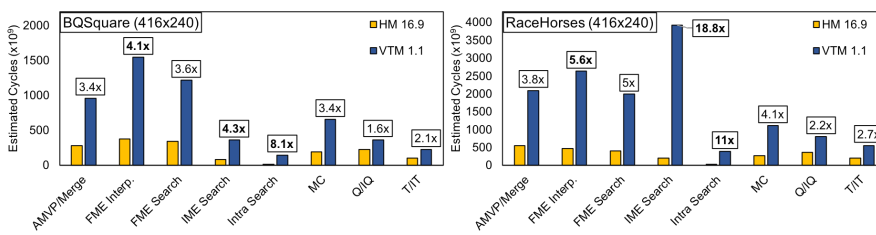


Figure 1.4: Cycle estimation of the main encoding steps for the HM and VTM encoders (obtained with the valgrind profiler) when encoding two sequences with distinct motion characteristics. (QP=22, 30 frames, Random Access)

as an optimization problem to minimize the cost J, considering rate R and distortion D of each coding mode, as shown in (1.2).

$$J = D + \lambda \cdot R \tag{1.2}$$

Ideally, R and D are obtained at the output of entropy coding and in-loop filter, respectively. This process is known as full RDO. However, it has a huge computational complexity, due to the high number of combinations of block partitioning and prediction modes of current video encoders. In practice, some coding modes are skipped from RD-cost calculation, and D is obtained from the prediction step, simplifying the calculation of J.

The following sections will present two important coding tools related to this metric: the block partitioning decision and the distortion computations (used to obtain D), highlighting important contributions that optimize each one.

1.4.1 Block partitioning decisions

As explained in Section 1.2.2, HEVC supports flexible coding structures to find the best encoding mode of different regions in the frame. This was necessary to achieve the superior coding performance of this standard, but it also conveys an optimization problem: finding the best CU, PU and TU partitioning in terms of RD-cost given all possible modes. The solution for this problem is not trivial, as there is no known algorithm capable of predicting the RD-cost efficiently.

The best solution can only be obtained by computing the actual cost of every possibility by applying the inter- and intra-frame prediction, transforms and quantization and their inverse counterparts, and entropy coding. This brute-force process, known as full RDO, requires a significant amount of computations. To mitigate this, several speed up heuristics are implemented in the current HM version, skipping some CU, PU and TU partitions depending on specific RD tests. However, real-time HEVC encoding remains an open problem despite the current software optimizations. Therefore, solutions that further reduce the complexity of the HEVC partitioning decisions are still being studied.

The main approach adopted in related works consists in early termination techniques for the CTU quadtree splitting process. When a certain condition is met, the current CU is not further split, eliminating the computations that would be spent to process its four sub-CUs. Early-terminating the quadtree nodes might remove important CU partitions that would increase the coding efficiency. Therefore, the challenge is designing efficient solutions to maximize the encoding time savings while minimizing the RD loss obtained from wrong early terminations.

In [27], Data Mining and Machine Learning techniques are used to train classification models that decide whether the current CU being processed should be further split or not. By analyzing encoding process in HM it was observed that RD-cost and best mode of the current CU are highly correlated with the final split decision. A Decision Tree was trained using these variables for each CU size. The reported results show a 36.7% encoding time reduction with a coding performance loss of 0.28% in BD-bitrate.

The work in [28] employs a classification-based CU split decision using Support Vector Machines (SVM). The classifiers are trained using encoding-domain variables obtained offline. The SVM model loading and prediction routines were implemented in the HM software to obtain encoding time savings and compression efficiency results. The results presented point to an average reduction of 48% in the encoding time, with a BD-bitrate loss of 0.48%.

In the following we present VLSI architectures to the most computing-intensive encoding steps. Note, however, that the algorithmic solutions presented in this section can be jointly applied with the hardware architectures discussed hereafter, further reducing the overall encoding time.

1.4.2 Distortion metrics

Three metrics are usually employed in reference implementations to estimate the distortion D between two blocks: Sum of Absolute Differences (SAD), Sum of Absolute Transformed Differences (SATD), and Sum of Squared Differences (SSD) (a.k.a. Sum of Squared Error - SSE). Although the most efficient metric to estimate distortion is SSD, as it approximates the PSNR better than other metrics, it is rarely found in practical hardware implementations because of its higher complexity. Hence, we focus on the most common metrics found in hardware implementations: SAD and SATD.

1.4.2.1 Sum of Absolute Differences (SAD)

SAD is calculated as given in (1.3), in which O is the original block, R is the reference/candidate block, m and n are the dimensions of both blocks in samples.

$$SAD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |O_{i,j} - R_{i,j}| \tag{1.3}$$

An example of SAD hardware architecture with four original (O) and four reference (R) input samples is shown in Fig. 1.5. It is composed by subtractors, absolute operators and an adder tree. An accumulator is placed at the output to calculate the final SAD for a block of Nx4 samples in N cycles. Different SAD architectures with 8-input, 16-input or other number of input samples and pipeline levels can be designed depending on the needed throughput and the available memory bandwidth of DPB and current frame buffers.

Recent works propose SAD architectures focusing on high throughput and low power dissipation. The adder tree can be implemented with different adder compressor topologies. Silveira et al. [29] propose a 8-input sample SAD architecture with 8-2 adder compressors implemented in different ways. Different absolute operators are proposed in [30]. The absolute operator designed with XNOR gates achieved the lower power dissipation compared to eight proposed solutions in [30] and the *abs* macrofunction implemented by the synthesis tool. Abreu et al. [31] proposes a Partial Distortion Elimination (PDE) optimization on SAD units which avoids the computation of candidates that will surely not be selected in the motion estimation search. Results of the solutions mentioned above are shown in Table 1.4. Power

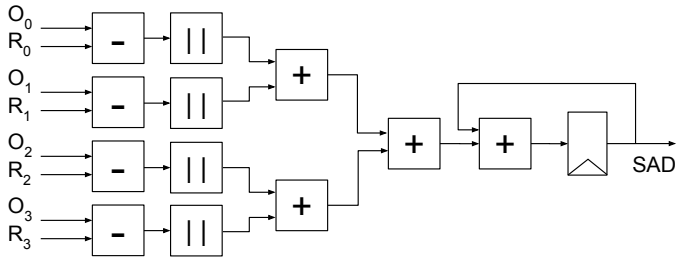


Figure 1.5: SAD hardware architecture with 4 input samples.

results in [29, 31] are averaged of difference video sequences, while those reported in [30] are obtained with one 1920x1080 video sequence. Gate count results are equivalent to 2-input NAND gate.

Table 1.4 Results of state-of-the-art SAD architectures

	Technology node	Maximum frequency (MHz)	Throughput (resolution @ frame rate)	Gate Count	Area (μm^2)	Total Power (μW)
[29]	45 nm	300	1920x1080 @ 30 fps	1838	1470	996.15
[30]	65 nm	133	1920x1080 @ 30 fps	1760	3665	488
[31]	65 nm	300	3840x2160 @ 30 fps	N.A.	N.A.	707.27

1.4.2.2 Sum of Absolute Transformed Differences

SATD calculation is given in equation 1.4, in which *HT(i,j)* is the Hadamard Transform (HT) of the residual block W, as shown in equation 1.5. HM encoder implements SATD with the option to use HT with three different Hadamard matrix sizes: 2x2, 4x4 and 8x8, as shown in equations 1.6, 1.7, and 1.8.

$$SATD_{n\times n} = \sum_{i,j} |HT_{n\times n}(i,j)| \tag{1.4}$$

$$HT_{n\times n} = H_{n\times n} \cdot W \cdot H^{T}_{n\times n}/(n/2) \tag{1.5}$$

$$H_{2\times 2} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{1.6}$$

$$H_{4\times 4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{1.7}$$

$$H_{8\times8} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \tag{1.8}$$

Two common architectural templates to implement SATD are found in the literature. Fig. 1.6a shows a parallel solution for SATD architecture employing a 2x2 HT. The subtractors calculate the difference between original (O) and reference/predicted (R) samples, that are further processed by a 2x2 HT and absolute operators. An adder tree calculate the final SATD value. For larger HT sizes, it is common to employ the semi-parallel solution (Fig. 1.6b) which use the separability property of HTs to design two 1-D HTs and a buffer in between the HTs. The buffer guarantees that second 1-D HT is applied after all coefficients are processed by the first 1-D HT.

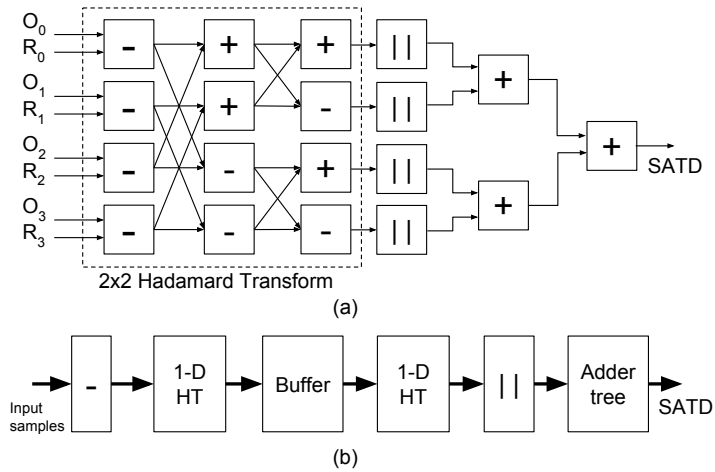


Figure 1.6: SATD hardware architecture. (a) Parallel solution using 2x2 HT. (b) Generic template of a semi-parallel HT solution.

In the same context of SAD, recent works propose SATD architectures focusing on high throughput and low power dissipation. Soares et al. [32] propose an approximate SATD architecture based on coefficient-prunning algorithm of 4x4 HT. The approximate computing strategy reduces power compared to precise SATD with negligible BD-PSNR reduction. Seidel et al. [32] propose different SATD architectures with HT sizes ranging from 4x4 to 32x32. It also evaluates two types of buffers (linear and transposition), but transposition buffer results in low operating frequency that is less useful for high resolution video coding. Silveira et al. [33] proposes an SATD architecture supporting multiple HT size (2x2, 4x4 and 8x8) and

adder compressors to reduce power. Since the calculation of HTs of different sizes share common operations, it is possible to reuse hardware of the architecture while supporting different HT sizes. The work in [34] proposes an SATD architecture with 8x8 HT using adder and subtractor compressors to reduce power in 14.4% compared to baseline SATD architecture. Monteiro et al. [35] demonstrates how previously computed HT can be re-used in the computation of larger blocks. A generic SATD hardware design based on this demonstration is also shown in [35]. Results of the solutions mentioned above are shown in Table 1.5.

Table 1.5 Results of state-of-the-art SATD architectures

	Matrix sizes	Technology node	Maximum frequency (MHz)	Throughput (resolution @ frame rate)	Gate Count	Area (μm^2)	Total Power (μW)
[36]	4×4	45 nm	650	N.A.	7553	N.A.	4535.6
[32]	4×4	45 nm	352	N.A.	N.A.	N.A.	283.27
[32]	4×4 3232x32	45 nm	264.5	N.A.	N.A.	N.A.	2424.4
[33]	2x2, 4x4, 8x8	45 nm	125	N.A.	N.A.	70321	7412.6
[34]	8x8	45 nm	333	1920x1080 @ 30 fps	20900	16729	5001.8
[35]	4x4, 8x8	45 nm	32	N.A.	N.A.	14978	467.8

1.4.3 Challenges on Rate Distortion Optimization for VVC encoder

Most of the solutions for fast HEVC partitioning decisions focus on the binary CU split problem, and that there is a growing trend of using learning-based techniques to build a solution. In VVC, this problem is harder to solve, because the Coding and Prediction Units are now part of the same structure, increasing the number of possible outcomes. CUs in VVC can assume five different types of splitting, one of which is the quad-split also supported in HEVC (see Fig. 1.3).

One possible approach is to train multi-class classifiers that are capable of predicting all of the possible partitioning modes supported in VVC. It is not possible to accurately predict if this will be an easier or harder problem compared with a binary framework, because it depends on how the attributes used for training will be distributed in each class (splitting mode).

A second approach would be relaxing the problem by putting all splitting modes in the same bag, so the decision to be made is still binary, i.e., either the current CU will not be split or it will be (in some of the 5 possible ways). This might be easier than the first approach, but it also has less potential of achieving significant time savings, because, when a split decision is made, every mode will have to be tested.

Some blocks will certainly be evaluated by SAD and SATD. State of the art SAD and SATD architectures target the performance requirements of HEVC encoder. The

performance requirements of SAD and SATD increased in VVC encoder compared to HEVC encoder, as shown in the time increases of IME and FME search in Section 1.3). Note that VVC and HM use fast ME algorithms, are we discuss in Section 1.5. VTM implementation uses different HT sizes to support the new block sizes.

Future solutions for RDO will need joint algorithm-architecture optimizations by combining fast block partitioning algorithms with high throughput hardware architectures employing intelligent reusing schemes, fast and low power arithmetic operators and newer technology nodes. New algorithms and architectures for inter-frame prediction are key to reduce the number of blocks evaluated by SAD and SATD, as we discuss in Section 1.5.

1.5 Inter-frame prediction

As discussed in Section 1.2, inter-frame prediction is composed by Motion Estimation (ME) and Motion Compensation (MC) steps. Since current video coding standards support fractional-precision motion vectors, ME stage is split into two stages: the Integer Motion Estimation (IME) and the Fractional Motion Estimation (FME). The following subsections detail both of these stages. MC design is simplified in video encoders since the blocks can be reconstructed and saved inside the ME module. Another important consideration is the high memory bandwidth required to fetch reference and current blocks from buffers to SAD accelerator inside the ME architecture. Therefore, we have also discussed about dedicated memory design for ME.

1.5.1 Integer Motion Estimation

Integer Motion Estimation (IME) is responsible for searching on reference previously encoded frames for similar blocks to the current block to be encoded. The search occurs inside a search area smaller than the frame itself, because image patterns tend to slightly displace from the area where they were in temporally close frames. This process generates an integer-precision motion vector (x,y) pointing to the upper-left corner of the best block found in the search. In the IME stage, the similarity between two blocks, i.e., the reference block and the current block, is calculated with the SAD metric in HEVC-compliant encoder softwares (such as HM and x265 [37]), as well as in VTM, in their default presets. SATD is used for FME and intra-frame prediction, as we will discuss in the following.

There are several fast block matching algorithms (BMA) to search for similar blocks in the IME stage, since it is not normatized by standards. They reduce the number of candidate blocks to be searched, compared to exhaustive search (called full search, which is not implemented in practice due to huge complexity). Fast BMAs do not lead to the optimal block inside the search area, but have acceptable RD-cost results. Two well-known algorithms are analyzed in this chapter, since they are found in HEVC and VVC encoders: Test Zone Search (TZS), which is used in HM and VTM, and Hexagon Search (HS), employed in x265. Both algorithms start their executions from a motion vector which was decided before the IME stage,

based on the resulting motion vectors of the temporally and spatially neighboring blocks of the one being executed. The analysis presented in this chapter considers the TZS implementation from the HM encoder software, and the HS from x265.

Both TZS and HS are divided into four interdependent stages. Such dependencies between stages complicate pipeline-based architectural implementations that may waste a high amount of additional cycles due to pipeline bubbles. These stages are explained in the following items.

- Search Vector Initialization: this stage is responsible for checking whether other motion vectors may be better than the initial chosen one to start the search in the next steps. This stage checks for the (0,0) vector and the resulting vector from the $2N \times 2N$ prediction of the same CU, to analyze if any of these are better than the initial motion vector.
- First Search: in TZS a diamond-shaped search is performed, starting from the position defined by the previous stage. The diamond patterns test four, eight or sixteen candidates around the center, depending on the distance between samples, which is incremented after each diamond step. In HS a six-point search in a hexagon shape is done around the center of the search. This stage ends when no better candidates have been found for three diamond steps (in TZS) or for one Hexagon iteration (in HS).
- Raster/Square Search: this stage searches through the entire search area (in TZS), only executing if the current best vector is in a specific distance from the center, or in an 8-point area, after the First Search (in HS).
- Refinement Search: considering TZS, the last stage works similarly to the First Search. The main difference here is that the algorithm may be iterated several times with the center being set to the current best center in a new iteration. The algorithm stops when no better candidates have been found in an entire iteration. In HS, this last stage checks whether a set of candidates decided before the IME has any better candidates.

To achieve real-time throughput with an IME architecture we have analyzed TZS and HS algorithms on default presets of HM and x265. Simulations were performed to obtain the number of 8×1 SAD calculations required to encode one second of five Full-High Definition (FHD) (1080p) sequences using the TZS and HS algorithms, as well as the quality difference (BD-BR) of HS when compared to TZS. The 8×1 SAD calculation is used to normalize the comparison for both algorithms, because of the different block sizes evaluated by SAD. The results are presented in Table 1.6.

The analysis presented in Table 1.6 show that HS requires about $4\times$ less 8×1 SAD computations when compared to TZS. The results are easily understandable, considering that the x265 is a faster encoding software and, therefore, needs to apply a faster BMA in its IME. However, when considering the quality results in terms of BD-BR, the HS obtains an average reduction of 0.63%, which indicates a considerable reduction in quality.

Therefore, similar analysis must be conducted in order to decide which BMA to implement in a hardware architecture, considering the demands of throughput and quality of a given application or service.

Table 1.6 Number of 8×1 SAD requests and BD-BR values for one second of different video sequences, comparing the TZS and HS BMAs.

	8×1 SAD calculations (1s)			
Video Sequence	TZS ($\times 10^6$)	HS ($\times 10^6$)	Reduction (%)	BD-BR (%)
BasketballDrive	14.11	2.39	83.1	-0.11
BQTerrace	6.36	2.24	64.8	-0.56
Cactus	6.37	1.81	71.6	-1.27
Kimono	5.24	1.14	78.2	-0.66
ParkScene	1.81	0.82	54.7	-0.55
Average	6.78	1.68	75.2	-0.63

The design of an IME architecture highly depends on the SAD architecture used within it. More specifically, the input width and the number of pipeline levels of the SAD architecture are two crucial factors to be considered in the design of the IME architecture. When using a high input width, SAD for high blocks can be calculated with less chunks accumulations. However, this requires a higher number of bits to be sent to the architecture in a given cycle, which may be an issue when dealing with memory communication. Additionally, the importance of the number of pipeline levels is due to the fact that the recent BMAs have several dependencies between the stages, which requires the pipeline to be emptied at every transition between stages, which may compromise the effective throughput of the architecture. For instance, when considering the HS BMA, the Square Refinement stage cannot begin until the last SAD chunk of the Hexagon stage is sent to the SAD accumulator, given that the last calculated candidate may be the best one and the algorithm may not switch to the Square Refinement, continuing for another iteration instead.

Along with the SAD architecture, some additional registers are an important part of generic IME architectures. Registers that store the best SAD are crucial so that this SAD value can be compared with the one currently being calculated, to check whether the current candidate has a smaller SAD value than the current best one.

A kernel (Control Unit) that generates the vector requisitions for the memory is also required. The values of the vectors are represented by x and y coordinates. Considering the two aforementioned BMAs, vectors are generated based on their current center. Therefore, it is recommended that registers for the current center (which may change with every single iteration) are used, so that the control unit may generate the current vector to be sent to the memory based on the current center and on the current candidate position, relative to the center. This control unit is also responsible for checking whether the new SAD chunk coming to the SAD register should be added to the current one or loaded as a new SAD. The number of accumulations depends on the size of the PU being encoded, as well as on the input width of the SAD architecture.

An IME architecture was implemented considering the Hexagon search algorithm. The full architecture is shown in Figure 1.7. The architecture considered an 8×1 SAD module (64 bits for each block chunk), due to the fact that, in the default preset of the x265 software, the smallest PU dimension is 8, and using larger SAD modules could lead to unused hardware when calculating SAD for smaller PUs. The

architecture did not employ pipeline in the SAD module, given that the Hexagon search contains several data dependencies between each of the Hexagon iterations, which would generate pipeline bubbles. The control block required to manage the new block, the accumulator register and the best SAD register are also shown in Fig. 1.7.

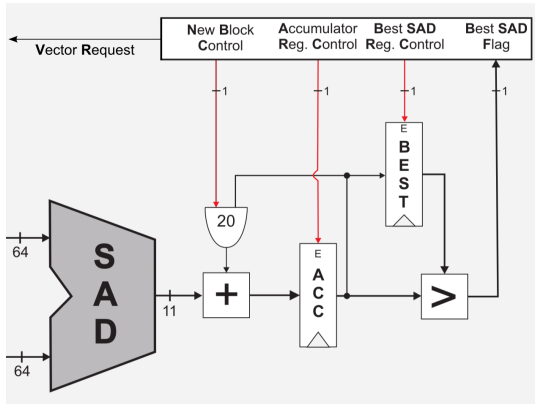


Figure 1.7: IME architecture and its Control Unit

We estimated the throughput required for the IME architecture to work in real-time based on the 8-byte requests from the x265 software, for the same Full HD video sequences from the previous analysis, for 30 frames of each video (equivalent to 1s of a 30fps video). In our analysis, the number of 8-byte requests of the SAD operation in the HS is equivalent to the required frequency. Table 1.7 shows the estimated frequency for the five videos, as well as the average case, which corresponds to our frequency goal.

Table 1.7 Frequency estimation (1080p@30fps) for different video sequences.

Video Sequence	Frequency (MHz)
BasketballDrive	561.61
BQTerrace	402.93
Cactus	441.19
Kimono	641.61
ParkScene	388.07
Average	487

The architecture was synthesized for ASIC, with ST 65 nm CMOS standard cells library, for the maximum frequency of 555 MHz. The area and power results of the IME architecture is shown in Table 1.8. The power results were obtained by applying real input vectors extracted from the SAD operation in the x265 software, to accurately estimate the dynamic power.

Table 1.8 Area and power results of the IME architecture.

Circuit Area		Power (μW)		
(μm^2)	kGates	Leakage	Dynamic	Total
20367.9	11.25	15.936	6950.626	6966.562

Based on the average frequency estimation of 487 MHz (from Table 1.7), and considering the maximum clock frequency of 555 MHz achieved, the architecture is able to achieve real-time throughput for 1080p videos at 30fps.

Even higher throughput values may be achieved by employing hardware-friendly BMAs, which ease the use of pipeline – by not introducing high dependencies between the stages – due to the increase on the frequency. Moreover, increasing the SAD input width may also lead to a higher throughput if the memory is able to dispatch block chunks in fewer cycles.

1.5.2 Fractional Motion Estimation

In some cases, the offset between an object in adjacent frames is smaller than a pixel. This is a product of high frame rates combined with low motion intensity. Since the motion vectors produced in the IME step are not capable of representing fractional offsets, a second search is required to include this possibility. This search is called Fractional Motion Estimation (FME) and is composed of two main steps: (i) an interpolation of fractional pixels, since the only ones available are in integer precision; and (ii) a block-matching search to find the direction in which the sub-pixel offset occurred. The second step generates a fractional motion vector that is added to the integer one, producing the final vector encoded in the bitstream.

In HEVC, the FME is two-fold: in the first search, half-pixel samples are interpolated, then a search is applied in the interpolated area to find the half-pixel motion vector; in the second search, a quarter-pixel area is interpolated using both the integer and half-precision pixels as input, and a new search is performed.

For a block with $W \times H$ pixels the area of both searches is limited to a $(2W + 1) \times (2H + 1)$ region surrounding the IME block, corresponding to eight fractional candidates. To generate this region in the first interpolation, a block of $(4+W+4) \times (4+H+4)$ integer samples must be fetched from memory (the four-pixel padding on each side is required to interpolate border samples), so memory access is also an important concern for designers.

After the half search is completed, another $(2W+1) \times (2H+1)$ search occurs in the region surrounding the best block found in the half search, now using quarter-interpolated samples. It is common to interpolate half and quarter samples before searching, so the quarter regions surrounding all possible half-pixel candidates are generated speculatively, yielding a $(4W+3) \times (4H+3)$ area. Fig. 1.8 shows a 2x2 integer block (shaded cells) and the surrounding half and quarter pixels that must be interpolated to perform both searches. Note that there are actually 4-pixel padding zones surrounding this block that were omitted for visual purposes.

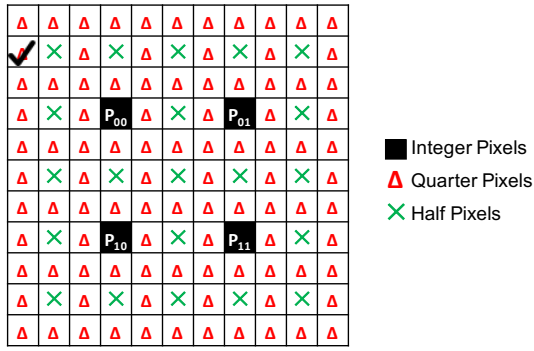


Figure 1.8: Interpolated area surrounding a 2x2 block (padding integer samples omitted).

The interpolation of fractional pixels in HEVC is computed using Finite Impulse Response (FIR) filters with different input sizes (usually referred to as number of taps) depending on the precision (half or quarter) and on the color channel being processed (luminance or chrominance). An N-tap FIR filter consists of multiplying the input samples by a set of N coefficients and adding up the partial results.

Luminance pixels are interpolated using 8-tap filters for half pixels and 7-tap filters for the quarter ones. Tab. 1.9 shows the coefficients used to interpolate luminance pixels for the quarter (1/4 and 3/4) and half (2/4) precisions.

Table 1.9 Luminance Interpolation Filter Types

Filter types	Filter coefficients							
1/4	-1	4	-10	58	17	5	1	
2/4	-1	4	-11	40	40	-11	4	-1
3/4	1	-5	17	58	-10	4	-1	
Integer sample index	-3	-2	-1	0	1	2	3	4

As pointed out in Section 1.3.1, interpolation is the most complex FME step due to its multiplications and large block sizes supported in the HEVC standard. Although hardware multipliers can be replaced by add/shift operations in this case (one of the terms is a constant coefficient), the throughput required to process fractional samples in real-time is enormous. Therefore, solutions that accelerate this process are demanded.

Diniz et. al. [38] propose an HEVC interpolation architecture that implements luma and chroma filters using configurable design techniques. The architecture processes 12 pixels in parallel and configures its datapaths to support different filter types. The configuration logic also adapts to the different execution scenarios that require interpolation: ME or MC.

A second approach by Diniz et. al. [39] employs dynamic reconfiguration in its design. Two engines are proposed to process luma and chroma samples with different throughput capabilities (up to 12 pixels in parallel) depending on the current data

path configuration. This work also features a component that predicts the number of filter calls based on observations from previous frames and knowledge obtained from analyses. The output of this prediction is used in conjunction with the current block size in an adaptive scheduling algorithm that selects the smallest amount of data paths capable of achieving the required throughput. It is capable of achieving a 2560x1600@30fps throughput with a frequency of 283 MHz in the luma engine and 184 MHz in the chroma one.

The work proposed in [40] employs adder compressors to replace regular ones in the add/shift-based multiplication part of the interpolation. Adder compressors are power-efficient parallel adders that can be used when the partial sums are not required, which is the case with interpolation. For luma filtering, a 7-2 adder compressor is employed, whereas a 8-2 compressor is used in the chroma interpolation. A second architecture discussed in the paper includes the final adder used for accumulation in the luma adder compressor unit. The presented results show that the proposed architecture consumes less power compared with competing solutions.

A multi-standard interpolation architecture capable of processing the HEVC, AVS and MPEG-2 interpolations is proposed in [41]. The authors found that the different standards shared similarities despite the distinct filters employed in each of them. This similarity was used to design luma and chroma architectures that deliver the interpolated pixels of a given standard depending on the input coefficients. Resource utilization was optimized by reusing operators common among standards. Using a 45 nm Standard Cell technology, the proposed architecture is capable of achieving a 7640x4320@60fps throughput in HEVC mode using a target frequency of 471 MHz.

Table 1.10 shows a summary of the references aforementioned. Note that the throughput results of every work only considers the MC requirements.

Table 1.10 Results of state-of-the-art interpolation filter architectures

	Platform	Max. frequency	Throughput resolution@fps[1]	Resource utilization	Area (μm^2)	Total power (μW)
[38]	TSCM 150 nm	312 MHz	3840x2160 @30fps	30,209 gates	N/A	N/A
[39]	Virtex-5 65 nm	283 MHz/ 184 MHz	2560x1600 @30fps	5,017 LUT+ 2,550 Register	N/A	89,000
[40]	Nangate 45 nm	707 MHz	N/A	11,201 gates	21,098	9,967
[41]	Nangate 45 nm	471 Mhz	7680x4320 @60fps	82,090 gates	65,508	33,790

[1] Considering Motion Compensation only

1.5.3 Dedicated memories for motion estimation

Video codecs introduce very high pressure on memory hierarchy, leading to undesirable energy and performance overhead [42]. Besides employing novel complex coding tools, an HEVC encoder requires a significant amount of data from the off-

/on-chip memories due to more memory intensive reference frames transmission for the prediction steps.

On average, the memory demand of HEVC is 2x-3x higher compared to the H.264/AVC [43]. Thus, high off-/on-chip memory bandwidth along with larger on-chip video memories (to support bigger resolutions) leads to increased energy consumption in the HEVC encoders. The memory bottleneck in HEVC encoders is related to the access to already processed (and reconstructed) video frames, called reference frames. In this aspect, up to 50% of off-/on-chip memory accesses are required for reference frames reading and writing leading to high memory-related energy consumption during HEVC encoding [42].

These aforementioned memory problems of HEVC tends to be aggravated when considering the next-generation of video encoders. In case of current VVC draft, several adopted coding tools may intricate the memory bottleneck. Initially, the novel strategy for CTUs partitioning, which is based on a quad-tree with more possibilities to be analyzed by encoder control, will lead higher memory demand to inter-frame prediction. Precisely, the motion estimation must be extensively executed to analyze more possibilities of motion vectors from several reference frames. Furthermore, refined FME precision (up to 1/16 pixel) will lead to extra on-chip memory to keep stored the generated fractional pixels. Still, another integrated coding tools may contribute to complicate the memory management for next-generation VVC encoders, like features of larger CTUs sizes and accurate motion vectors resolutions [20].

Therefore, there is a strong need of dedicated memory infrastructure (in both on-chip and off-chip perspective) and application-driven memory management to reduce the bandwidth requirements of current (HEVC) and next-generation (VVC) video coding applications.

To provide on-chip data storage support in ME architectures in HEVC encoders, several works proposed on-chip video memory architectures along with memory management strategies [42, 44, 45]. The main goal is to reduce memory energy consumption by adopting novel memory organizations (combined private/shared memories with mixed cache/scratchpad design) and emerging memory technologies (SRAM along with MRAM/STT-RAM[10] cells).

1.6 Intra-frame prediction

Intra-frame prediction is included in the latest video encoders such as H.264/AVC, HEVC and the upcoming VVC. Intra prediction is employed to exploit pixel correlation within a video frame. The intra prediction is performed in the spatial domain by using neighbor samples already coded and reconstructed to predict the current block. The neighbor samples are localized in regions left, up, left-down and up-right from the current block and they can be interpolated in different ways to generate the predicted block.

The number of possible ways to perform the interpolation of the neighbor samples, called intra prediction modes, is increasing with the evolution of the video

[10]Magnetic/Spin Tranfer-Torque RAM memory technology

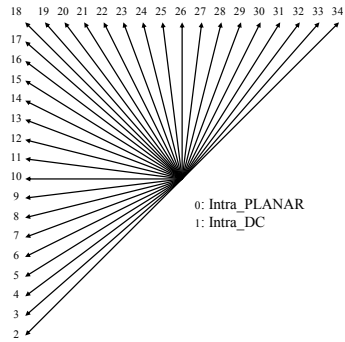


Figure 1.9: Intra prediction modes in H.265/HEVC [46].

coding standards, since they need to improve compression rates targeting high resolution videos and immersive video. Whereas in the H.264/AVC there were nine intra prediction modes, HEVC increase this number to 35, and for the VVC this number is currently in 67 modes. Figure 1.9 shows the available intra prediction modes in the HEVC.

The high number of intra prediction modes together with all possible block sizes available in current standards makes the intra prediction mode selection one of the main challenges of the video encoding. The following subsections present the intra prediction mode selection and a hardware implementation of the intra prediction for the HEVC standard.

1.6.1 Intra prediction mode decision in H.265/HEVC

The main challenge of the intra prediction in HEVC is to choose which mode will be used for each PU. RDO (Section 1.4) is used to deal with this decision, however, it can not evaluate all possible intra prediction modes, since it would be very time consuming. The solution to deal with this problem evaluates only a subset of the total number of modes. In HM, the intra prediciton mode selection is performed using a three-step mechanism composed by Rough Mode Decision (RMD), Most Probable Mode (MPM), and RDO. Figure 1.10 shows this three-step mechanism.

RMD goal is to reduce computational complexity of RDO. In the RMD, the predicted PU is generated for all 35 modes and SATD is calculated to each mode. Once all 35 modes were evaluated for the current PU, the **N** modes with the lowest RMD costs are inserted into a candidate list. The candidate list size varies according the PU size: 8, 8, 3, 3 and 3 are the number of modes in the candidate list for the 4×4, 8×8, 16×16, 32×32, and 64×64 PU sizes, respectively. After RMD, the MPM step exploits the correlation of the current PU with two neighbor PUs. Based on the intra prediction modes of these two PUs, three modes that are most probable to be selected are inserted into the candidate list. This way, modes that were discarded in the RMD process can be inserted into the list by the MPM evaluation.

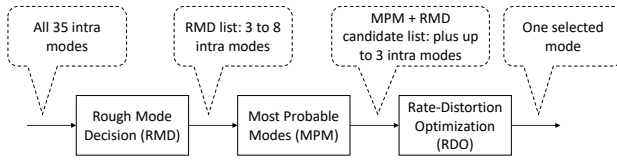


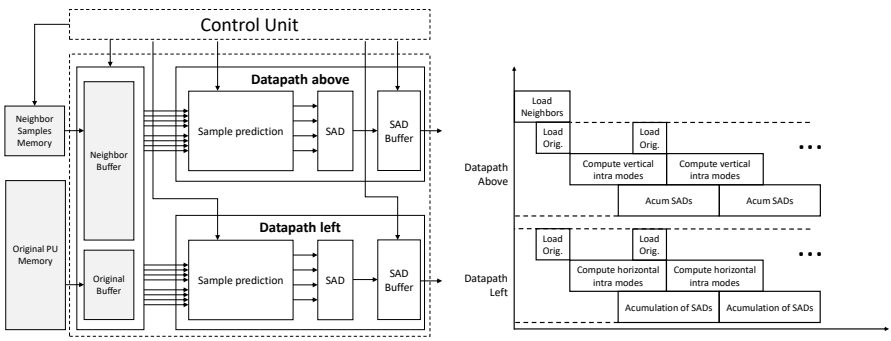
Figure 1.10: Intra prediction mode decision in HEVC.

Even with this simplification the intra prediction mode selection is still a high computational intensive task in HEVC encoders, since the RDO process is performed several times in the CTU to select the final mode. Therefore, several works, such as [46], focus on reducing the complexity of the intra mode selection process.

1.6.2 Hardware architecture for the HEVC intra prediction

he hardware architecture for intra-frame prediction is based on the work proposed in [47]. The architecture supports all intra prediction modes, provides high throughput to real time processing of high resolution videos, and reduces memory accesses. Figure 1.11a shows a high-level block diagram of the designed architecture.

To achieve high throughput, the architecture is divided into two datapaths (called datapath above and datapath left). The datapath above processes all vertical modes (17 directions) while the datapath left processes all horizontal modes (17 directions). Mode 3 (45°) is performed in both datapaths to keep the architecture regularity. Intra prediction modes with equal directional angles use the same constant values to perform the sample prediction and the memory addresses to access the neighbor pixels are the same. This strategy simplifies the control. Each datapath processes four samples per clock cycle, so eight samples in parallel are delivered. The complete architecture is composed by 9 pipeline stages including the sample prediction and SAD calculation. Two on-chip memories (neighbor and original memory) were designed to locally store the input samples for intra prediction.



(a) High-level block diagram

(b) Pipeline diagram

Figure 1.11: Intra-frame prediction hardware architecture

First, the fetches all neighbor samples, once for each PU size, from the neighbor memory to neighbor buffer. With all input samples available after the loading step, the pipeline strategy improves the throughput and the architecture usage, since after the latency of loading all samples the architecture will always have valid input data to process.Figure 1.11a shows the timing schedule of the designed architecture. The time needed to load these neighbour samples varies from 9 to 129 clock cycles depending on the target PU size. The loading of the four original samples for each data path takes only one clock cycle. Finally, it is possible to process all modes for the PU being predicted, to calculate and to store the SAD partial values. The architecture takes 73,682 clock cycles to process an entire CTU, i.e, all possible modes for all possible PU sizes. Table 1.11 presents the results of intra-frame prediction architecture.

Table 1.11 Synthesis results

Technology node	Gate count	Max. Freq. (MHz)	Throughput (resolution @ frame rate)
65 nm	36,734	500	1920×1080 @ 13 fps

1.6.3 Challenges on intra-frame prediction architecture design for VVC encoder

VVC standard employs a higher number of intra prediction modes and block sizes. Several challenges arise when designing hardware architectures for the intra prediction in VVC, such as how to design energy efficient architectures to support more intra prediciton modes, how to increase parallelism to allow high throughput architectures, and how to design dedicated memory systems to feed high parallel architectures with constant data flow.

1.7 Transforms

The residual blocks (difference between current blocks and predicted blocks) are processed by block transforms in current standards. HEVC defines integer block transforms based on Discrete Cosine Transform (DCT) for the following TU sizes: 4×4, 8×8, 16×16, and 32×32. An alternative 4×4 transform based on Discrete Sine Transform (DST) is applied to only to luma samples of 4×4 intra-predicted blocks. The transform calculation is similar to the one already shown in equation 1.5 for HTs in Section 1.4.2.1. Similar to HT, 2D DCT can be calculated by two 1D DCT transforms with a transposition operation between the two calculations. The 4×4 DCT transform matrix is shown in equation 1.9.

$$C_{4\times4} = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix} \tag{1.9}$$

State of the art DCT architectures are proposed targeting HEVC. Conceicao et al. [48] propose a 16×16 2D DCT. Hardware minimization strategies were used to simplify hardware architectures, e.g. operation reordering, factoring, convert multiplications to shift-add operations, and sharing of common sub-expressions. Goebel et al. [49] propose 1D DCT hardware architecture for all transform sizes presenting a constant throughput of 32 coefficients per cycle independently of the transform sizes combination. Bonatto et al. [50] propose 1D DCT hardware architecture supporting all transform sizes. It employs operand isolation technique to reduce power based on a statistic analysis on DCT block sizes evaluated in HM software. Results of state-of-the-art works are summarized in Table 1.12.

Table 1.12 Results of state-of-the-art DCT architectures

	Dimension	Transform sizes	Technology node	Maximum frequency (MHz)	Throughput (resolution @ frame rate)	Gate Count	Total Power (mW)
[48]	2D	16×16	65 nm	742.02	7680×4320 @ 238 fps	14954	135.4
[49]	1D	All	45 nm	50	7680×4320 @ 32 fps	97300	24.2
[50]	1D	All	65 nm	54.56	3840×2160 @ 32 fps	124.8	30.56

1.7.1 Challenges of transforms architectures design for VVC encoder

The challenge of designing transform hardware architectures for VVC rely on supporting more DCT sizes than in HEVC, ranging from 2×2 to 64×64. In addition to that, VVC unifies transform and prediction in the same units with same block size, which leads to the transforms to handle non-square blocks such as 8×4, 4×8 and so on. It results in an increase on the performance requirements, especially to large transform sizes (e.g. 32×32 and 64×64). To handle to such requirement increase, higher throughput hardware architecture design of DCT is desired.

1.8 In-loop filter

In-loop filter is applied to reconstructed samples before delivering them to DPB, to improve the RD results of inter-frame prediction. In HEVC, it is by Deblocking Filter (DF), that is applied only to boundary samples of blocks larger than 4×4 samples to reduce blocking artifacts caused by strong quantization. Sample Adaptive Offset (SAO) filter is applied after DF to all samples of the image, by adding an offset of some pixels classified based on pixel intensity and edge properties. A few hardware solutions are found for DF and SAO, e.g. [51] and [52]. Adaptive Loop Filter (ALF) was considered to be included in HEVC but was removed prior to finalization of the standard. In-loop filter in VVC includes DF, SAO and ALF again. Since in

our analysis (Section 1.3) these filters spend together less than 1% of encoding time of VTM, we foresee that their performance requirements will easily be targeted by current software/hardware solutions.

1.9 Entropy coding

The entropy coding is the final encoding step hybrid video-coding standards, such as H.264/AVC, HEVC, and VVC [11, 12, 18, 19]. All data previously generated by the video encoder process is categorized into Syntax Elements (SE) that undergo the entropy encoding process. In HEVC, the CABAC (Context-Adaptive Binary Arithmetic Encoder) algorithm is used to perform this task.

CABAC is an Binary Arithmetic Coding (BAC) algorithm [53], which means that the coding decisions are binary (i.e., the choice of the algorithm is always between two possibilities). The advantage of using a BAC comes from the fact that the alphabet of symbols is reduced to 2, simplifying the design of solutions compared with a Multiple-Symbol AC [54].

The control variables of this process are Range, which corresponds to the interval in which the overall probabilities distributions for all encoded symbols are represented, and Low, which is the smallest possible value.

CABAC is composed of three steps: Binarization, Context Modeling, and Binary Arithmetic Encoder [54]. The details of these blocks are presented next.

During Binarization, the generated SEs are converted to specific binary representations, and each bit of this new representation is called a bin. A bin can be of three different types: regular, bypass, and terminate. Most encoded bins fall into the regular and bypass categories.

Regular bins are divided into two groups: the Most Probable Symbol (MPS) or the Least Probable Symbol (LPS). This is the only type of bin that undergoes the Context Modeling step. Bypass bins, on the other hand, are categorized as their value (i.e., '0' or '1'), because each value is randomly distributed throughout the encoding flow. Thus, such bins do not require the Context Modeling step.

The Context Modeling updates the occurrence probabilities for regular MPS and regular LPS bins. Every time a regular bin occurs, the likelihood of occurrence of the next MPS or LPS for the same SE is updated. The new probability is indicated by a variable called State, which is used in a Look-up Table fashion to decide the occurrence probability of the LPS bin (and by consequence the MPS bin, as will be further explained).

The Binary Arithmetic Encoder (BAE) is the central part of the CABAC block. To avoid the multiplications involved in the AC step, CABAC utilizes a pre-computed Range for the LPS bin (i.e., the rLPS variable), which is addressed by the State variable coming from the Context Modeling [54]. Hence, the update of the remaining control variables follows (1.10) for regular bins. For bypass bins, the Range is not updated (since each bypass value is equiprobable), and the control variables follow (1.11). The Range and Low variables may be re-normalized if they fall below or above specific values [12].

$$\begin{cases} rMPS = Range - rLPS \\ Range = rMPS \quad and \quad Low = Low, \quad when \quad bin = MPS \\ Range = rLPS \quad and \quad Low = Low + rMPS, \quad when \quad bin = LPS \end{cases} \tag{1.10}$$

$$\begin{cases} Range = Range \\ Low = Low, \quad when \quad bin = 0 \\ Low = Low + Range, \quad when \quad bin = 1 \end{cases} \tag{1.11}$$

1.9.1 Upcoming challenges related to entropy encoding

As already mentioned, VVC will possibly make use of the CABAC as entropy encoding algorithm. The usage of CABAC implies that the BAE has to be the same, whereas the Binarization and Context Modeling may change in case new SEs are added to the new standard, or different types of binarization are chosen for an already existing SE. The BAE is the critical part in terms of throughput bottleneck [55].

All recent CABAC alternatives propose multi-bin processing, where a multi-core structure is a baseline for the BAE block, so the other blocks have to deliver the minimum required quantity of bins that the BAE can process per clock cycle. As reported in [56, 55], a four-core BAE offers the best trade-off concerning the achievement in terms of throughput.

The first alternative consists in decreasing the critical path. The basic approach is to use a pipeline structure within the BAE. A possible four-stage pipeline architecture is depicted in Fig. 1.12. The first stage corresponds to the rLPS pre-selection, the second to the Range update, followed by the Low (and auxiliary variable OB) update, and finally the bitstream generation in the final stage [56].

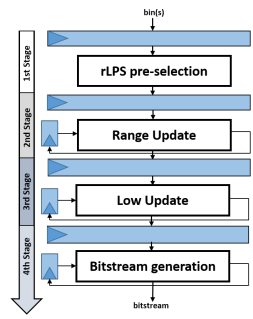


Figure 1.12: Baseline 4-stage pipeline approach for the BAE

The second option is to reduce the logic of the Range update. It is possible to move the rLPS variable re-normalization to the first stage and process it speculatively, as proposed in [57], increasing maximum frequency by 9.6%. This solution was named pre-normalization rLPS (PNrLPS)

Another alternative is to use heterogeneous cores that can process only a particular type of bin, instead of all of them. The approach presented in [55], named

LH rLPS, uses a structure where seven distinct cores are employed in the four-stage pipeline BAE design. The designed cores are capable of processing either LPS or MPS bins. In doing so, some cores require less hardware resources and a reduced critical path. This technique improves frequency by 29% in frequency [55].

In [57], a solution that completely separates the flow of regular and bypass bins, named Bypass-Bin Splitting (BPBS), is presented. Hence, the merge of the streams occurs only at the Low update stage, which now has five cores, each one able to process a single regular bin of any type or a single bypass bin. This solution increases the bins/cycle throughput in around 9.5%.

This idea was improved by processing more than one bypass bin per Low core in [58, 59]. The proposed scheme processes two bypass bins per core in parallel, with an average increase of 12.76% in throughput.

A final remark is that the techniques presented herein can be jointly applied to achieve even better results, which is depicted in Fig. 1.13. The usage of all techniques combined leads to an 8-stage pipeline structure, where the first stage is the rLPS pre-selection [56] integrated with PN rLPS [57]. The first and the second stages use the LH rLPS technique [55], necessary only for regular bins, since the usage of BPBS [57] separate the flows, which is shown in the same figure. Finally, the MBBS [59] technique is integrated into the fifth, sixth, seventh and eight stages.

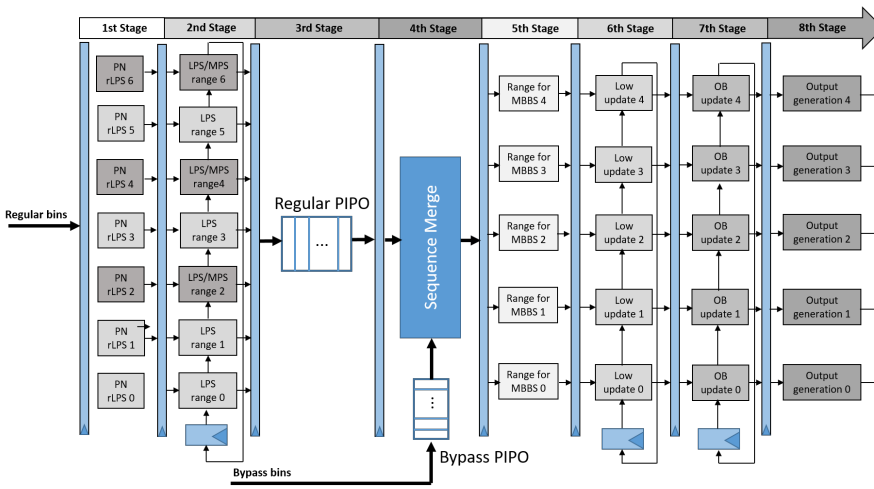


Figure 1.13: BAE architecture with all related techniques integrated.

1.10 Conclusions

TO DO.

These types of challenges on implementing high throughput hardware modules need to be tackled by industry and academic communities to make the future VVC video coding standard usable for a wide range of applications.

References

[1] Vanne J, Viitanen M, Hamalainen TD, et al. Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs. IEEE Transactions on Circuits and Systems for Video Technology. 2012 Dec;22(12):1885–1898.

[2] CISCO. Cisco visual networking index: Forecast and methodology 2017-2022; 2018.

[3] Khan MUK, Shafique M, Grellert M, et al. Hardware-software collaborative complexity reduction scheme for the emerging HEVC intra encoder. In: Proceedings of the Conference on Design, Automation and Test in Europe. EDA Consortium; 2013. p. 125–128.

[4] Shafique M, Henkel J. Hardware/software architectures for low-power embedded multimedia systems. Springer Science & Business Media; 2011.

[5] Zatt B, Shafique M, Henkel J, et al. 3D video coding for embedded devices. Springer; 2013.

[6] ITU-T. Recommendation H.261, version 1. Video Codec for Audiovisual Services at 64 kbit/s; 1988.

[7] ISO/IEC. 11172-2 - MPEG 1. Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/sPart 2: Video; 1993.

[8] ITU-T/ISO/IEC. Recommendation H.262 and ISO/IEC 13818-2 (MPEG 2 Video). Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video; 1994.

[9] ITU-T. ITU-T Recommendation H.263. Video Coding for Low Bit Rate Communication; 1995.

[10] ISO/IEC. Coding of Audio-Visual Objects - Part 2: Visual, ISO/IEC 14496-2 (MPEG-4 Visual version 1); 1999.

[11] ITU-T/ISO/IEC. ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC). Advanced video coding for generic audiovisual services; 2003.

[12] ITU-T/ISO/IEC. ITU-T Recommendation H.265 and ISO/IEC 23008-2. High Efficiency Video Coding; 2013.

[13] Sullivan GJ, Ohm JR, Han WJ, et al. Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on Circuits and Systems for Video Technology. 2012 Dec;22(12):1649–1668.

[14] HEVC Test Model (HM) v. 16.9;. http://hevc.hhi.fraunhofer.de.

[15] for Open Media (AOM) A. AV1 Bitstream & Decoding Process Specification, version 1.0.0, https://aomedia.org; 2018.

[16] Nguyen T, Marpe D. Future Video Coding Technologies: A Performance Evaluation of AV1, JEM, VP9, and HM. In: 2018 Picture Coding Symposium (PCS); 2018. p. 31–35.

[17] Laude T, Adhisantoso YG, Voges J, et al. A Comparison of JEM and AV1 with HEVC: Coding Tools, Coding Efficiency and Complexity. In: 2018 Picture Coding Symposium (PCS); 2018. p. 36–40.

[18] ITU-T/ISO/IEC. Joint Call for Proposals on Video Compression with Capability beyond HEVC. Joint Video Exploration Team (JVET); 2017.

[19] ITU-T. Requirements for Future Video Coding (FVC). ITU-T SG16/Q6 VCEG 56th meeting, Torino, Italy, July 2017, Doc. VCEG-BD03 (available at http://wftp3.itu.int/av-arch/video- site/1707_Tor);.

[20] Bross B. Versatile Video Coding (Draft 1). 10th Meeting of Joint Video Experts Team (JVET): San Diego, US, 1020 Apr. 2018, Doc. JVET-J1001-v2 (available at http://phenix.it-sudparis.eu/jvet/);.

[21] VVC Test Model (VTM) v. 1.1;. http://jvet.hhi.fraunhofer.de.

[22] Chen J, Alshina E. Algorithm description for Versatile Video Coding and Test Model 1 (VTM 1). 10th Meeting: San Diego, US, 1020 Apr. 2018, Doc. JVET-J1001-v2 (available at http://phenix.it-sudparis.eu/jvet/);.

[23] Bossen F. Common test conditions and software configurations. In: JCT-VC document no. L1100; 2013. .

[24] Bjontegaard G. Calculation of Average PSNR Differences between RD-curves. Video Coding Experts Group from ITU-T; 2001. VCEG-M33.

[25] Nethercote N, Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation. In: ACM Sigplan notices. vol. 42. ACM; 2007. p. 89–100.

[26] Sullivan GJ, Wiegand T. Rate-distortion optimization for video compression. IEEE Signal Processing Magazine. 1998 Nov;15(6):74–90.

[27] Correa G, Assuncao PA, Agostini LV, et al. Fast HEVC encoding decisions using data mining. IEEE transactions on circuits and systems for video technology. 2015;25(4):660–673.

[28] Grellert M, Zatt B, Bampi S, et al. Fast Coding Unit Partition Decision for HEVC Using Support Vector Machines. IEEE Transactions on Circuits and Systems for Video Technology. 2018;.

[29] Silveira B, Paim G, Abreu B, et al. Power-Efficient Sum of Absolute Differences Hardware Architecture Using Adder Compressors for Integer Motion Estimation Design. IEEE Transactions on Circuits and Systems I: Regular Papers. 2017 Dec;64(12):3126–3137.

[30] Abreu B, Paim G, Grellert M, et al. Exploiting absolute arithmetic for power-efficient sum of absolute differences. In: 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS); 2017. p. 522–525.

[31] Abreu B, Santana G, Grellert M, et al. Exploiting Partial Distortion Elimination in the Sum of Absolute Differences for Energy-Efficient HEVC Integer Motion Estimation. In: 2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI); 2018. p. 1–6.

[32] Seidel I, Brscher AB, Gntzel JL, et al. Energy-efficient SATD for beyond HEVC. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS); 2016. p. 802–805.

[33] Silveira B, Ferreira R, Paim G, et al. Low power SATD architecture employing multiple sizes Hadamard Transforms and adder compressors. In: 2017 15th IEEE International New Circuits and Systems Conference (NEWCAS); 2017. p. 277–280.

[34] Silveira B, Abreu B, Paim G, et al. Using adder and subtractor compressors to sum of absolute transformed differences architecture for low-power video encoding. In: 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS); 2017. p. 409–493.

[35] Monteiro M, Seidel I, Gntzel JL. On the calculation reuse in hadamard-based SATD. In: 2018 IEEE 9th Latin American Symposium on Circuits Systems (LASCAS); 2018. p. 1–4.

[36] Soares LB, Diniz CM, da Costa EAC, et al. A novel pruned-based algorithm for energy-efficient SATD operation in the HEVC coding. In: 2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI); 2016. p. 1–6.

[37] HEVC x265 Encoder;. https://bitbucket.org/multicoreware/x265/.

[38] Diniz CM, Shafique M, Bampi S, et al. High-throughput interpolation hardware architecture with coarse-grained reconfigurable datapaths for HEVC. In: 2013 IEEE International Conference on Image Processing; 2013. p. 2091–2095.

[39] Diniz CM, Shafique M, Bampi S, et al. A Reconfigurable Hardware Architecture for Fractional Pixel Interpolation in High Efficiency Video Coding. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2015 Feb;34(2):238–251.

[40] Diniz CM, Fonseca MB, da Costa EAC, et al. Evaluating the use of adder compressors for power-efficient HEVC interpolation filter architecture. Analog Integrated Circuits and Signal Processing. 2016;89(1):111–120.

[41] Penny W, Goebel J, Paim G, et al. High-throughput and power-efficient hardware design for a multiple video coding standard sample interpolator. Journal of Real-Time Image Processing. 2018;p. 1–18.

[42] Sampaio FM, Zatt B, Shafique M, et al. Hybrid Scratchpad Video Memory Architecture for Energy-Efficient Parallel HEVC. IEEE Transactions on Circuits and Systems for Video Technology. 2018;p. 1–1.

[43] Sampaio F, Shafique M, Zatt B, et al. dSVM: Energy-efficient distributed Scratchpad Video Memory Architecture for the next-generation High Efficiency Video Coding. In: 2014 Design, Automation Test in Europe Conference Exhibition (DATE); 2014. p. 1–6.

[44] Sampaio F, Shafique M, Zatt B, et al. Approximation-aware Multi-Level Cells STT-RAM cache architecture. In: 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES); 2015. p. 79–88.

[45] Sampaio F, Shafique M, Zatt B, et al. Content-driven memory pressure balancing and video memory power management for parallel High Efficiency Video Coding. In: 2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED); 2014. p. 153–158.

[46] Palomino D, Cavichioli E, Susin A, et al. Fast HEVC intra mode decision algorithm based on new evaluation order in the Coding Tree Block. In: 2013 Picture Coding Symposium (PCS); 2013. p. 209–212.

[47] Palomino D, Sampaio F, Agostini L, et al. A memory aware and multiplierless VLSI architecture for the complete Intra Prediction of the HEVC emerging standard. In: 2012 19th IEEE International Conference on Image Processing; 2012. p. 201–204.

[48] Conceição R, Souza J, Jeske R, et al. Low-cost and high throughput hardware design for the HEVC 16x16 2-D DCT Transform. Journal of Integrated Circuits and Systems. 2014;9(1):25–35.

[49] Goebel J, Paim G, Agostini L, et al. An HEVC multi-size DCT hardware with constant throughput and supporting heterogeneous CUs. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS); 2016. p. 2202–2205.

[50] Bonatto LVM, Ramos FLL, Zatt B, et al. Low-power multi-size HEVC DCT architecture proposal for QFHD video processing. In: Proceedings of the 30th Symposium on Integrated Circuits and Systems Design: Chip on the Sands. ACM; 2017. p. 41–46.

[51] Diniz CM, Shafique M, Dalcin FV, et al. A deblocking filter hardware architecture for the high efficiency video coding standard. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE); 2015. p. 1509–1514.

[52] Rediess F, Conceio R, Zatt B, et al. Sample adaptive offset filter hardware design for HEVC encoder. In: 2014 IEEE Visual Communications and Image Processing Conference; 2014. p. 299–302.

[53] Moffat A, Neal RM, Witten IH. Arithmetic coding revisited. In: IEEE Data Compression Conference (DCC); 1995. p. 202–211.

[54] Marpe D, Schwarz H, Wiegand T. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. IEEE Transactions on Circuits and Systems for Video Technology. 2003 Jul;13(7):620–636.

[55] Zhou D, Zhou J, Fei W, et al. Ultra-high-throughput VLSI architecture of H.265/HEVC CABAC encoder for UHDTV applications. IEEE Transactions on Circuits and Systems for Video Technology. 2015 Mar;25(3):497–507.

[56] Fei W, Zhou D, Goto S. 1 Gbin/s CABAC encoder for H.264/AVC. In: European Signal Processing Conference (EUSIPCO); 2011. p. 1524–1528.

[57] Zhou J, Zhou D, Fei W, et al. A high-performance CABAC encoder architecture for HEVC and H.264/AVC. In: IEEE International Conference on Image Processing (ICIP); 2011. p. 1568–1572.

[58] Ramos FLL, Zatt B, Porto M, et al. Novel Multiple Bypass Bins Scheme for Low-power UHD Video Processing HEVC Binary Arithmetic Encoder Architecture. In: Symposium on Integrated Circuits and Systems Design (SBCCI); 2017. p. 47–52.

[59] Ramos FLL, Zatt B, Porto M, et al. High-Throughput Binary Arithmetic Encoder using Multiple-Bypass Bins Processing for HEVC CABAC. In: IEEE International Symposium on Circuits and Systems (ISCAS); 2018. p. 1–5.