Cost and Coding Efficient Motion Estimation Design Considerations for High Efficiency Video Coding (HEVC) Standard

Mahmut E. Sinangil, *Member, IEEE*, Vivienne Sze, *Member, IEEE*, Minhua Zhou, Senior *Member, IEEE*, Anantha P. Chandrakasan, *Fellow, IEEE*

***Abstract*— This paper focuses on motion estimation engine design in future high-efficiency video coding (HEVC) encoders. First, a methodology is explained to analyze hardware implementation cost in terms of hardware area, memory size and memory bandwidth for various possible motion estimation engine designs. For 11 different configurations, hardware cost as well as the coding efficiency are quantified and are compared through a graphical analysis to make design decisions. It has been shown that using smaller block sizes (e.g. 4×4) imposes significantly larger hardware requirements at the expense of modest improvements in coding efficiency. Secondly, based on the analysis on various configurations, one configuration is chosen and algorithm improvements are presented to further reduce hardware implementation cost of the selected configuration. Overall, the proposed changes provide 56x on-chip bandwidth, 151x off-chip bandwidth, 4.3x core area and 4.5x on-chip memory area savings when compared to the hardware implementation of the HM-3.0 design.**

***Index Terms*— Hardware implementation cost, HEVC, motion estimation, search algorithm.**

I. Introduction

DURING the past decade, the amount of video content available on the Internet has grown significantly. With the introduction of 3G/4G mobile broadband technology, consumers can access this content from their mobile devices. Hence, by 2015, 70% of the mobile data traffic is expected to be attributed to video content [1]. In this context, standards with high coding efficiency are crucial for lowering transmission and storage costs.

Recent video coding standards such as AVC/H.264 provided significant coding efficiency gain over their predecessors. For example, AVC/H.264 provided 50% coding efficiency gain over MPEG-2 [2]. However, this improvement comes at the expense of higher hardware cost due to more complex coding tools as AVC/H.264 has 4× more hardware complexity with respect to MPEG-2 [2]. The trend for increasing hardware complexity over the years can be seen in Fig. 1 where relative complexity of the video core of a mobile applications processor is plotted over the years from 2004 to 2020 [3]. This figure reflects the increased complexity due to

- more advanced video coding standards and
- the necessity to employ a more dedicated hardware

to meet performance requirements.

By the year 2020, the complexity of a video core is expected to be 10× larger than today's demands [3]. Consequently, it is very critical to consider the hardware implementation cost in terms of hardware area, memory area (based on the capacity and the type of the memory) and memory bandwidth (rate at which data is accessed) of video codecs especially for mobile devices.

A. High-Efficiency Video Coding (HEVC)

High-Efficiency Video Coding (HEVC) is a new video compression standard being standardized by the JCT-VC (joint collaborative team on video coding) established by ISO/IEO MPEG and ITU-T [4]. HEVC achieves 50% coding gain over AVC/H.264 High Profile [5]. For this purpose, several coding efficiency enhancement tools have been adopted to this new standard. Table I provides a comparison between some of the tools used in AVC/H.264 and HEVC standards.

One of the main differences of HEVC from its predecessor AVC/H.264 is the adoption of coding quad-tree structure to provide a modular coding structure. In HEVC a frame is divided into largest coding unit (LCU) and an LCU is further divided into coding units (CU) in a quad-tree structure. LCU size can be as large as 64×64 pixels and smallest coding unit (SCU) size can be as small as 8×8. This allows the selection of a different coding structure based on various factors such as

Funding is provided by Texas Instruments.

M. E. Sinangil is with Nvidia, Bedford, MA 01730 USA (e-mail: msinangil@nvidia.com).

V. Sze and M. Zhou are with Texas Instruments, Dallas, TX 75243 USA (e-mail: sze@ti.com and zhou@ti.com).

A. P. Chandrakasan is with the Electrical Engineering and Computer Science Department, Massachusetts Institute of Technology, Cambridge, MA 02139 USA, (e-mail: anantha@mtl.mit.edu).

input video resolutions and other properties of a video sequence.

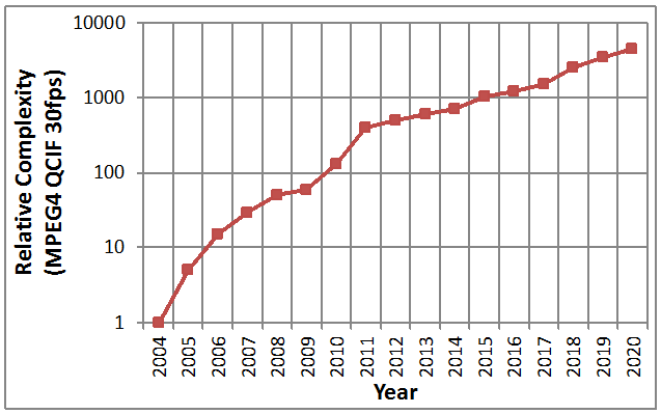


Fig. 1. Relative complexity of video core over the years for a mobile applications processor [3]. From 2012 to 2020, video core complexity is expected to increase by 10×.

Tool	AVC/H.264	HEVC
Coding Quad-Tree Structure	No	Yes
Largest Coding Unit Size	16×16	64×64
Asymmetric Motion Partitions	No	Yes
Inter-prediction Merge Mode	No	Yes
Transform Size	4×4 to 8×8	4×4 to 32×32
Intra-prediction Angular Directions	8 directions	33 directions

Table I. Comparison of some tools in AVC/H.264 High Profile and next generation video standard, HEVC. More complex HEVC tools require more complex hardware.

If a CU is not divided into smaller CUs, it is predicted with one of several prediction unit (PU) types. Either inter-prediction or intra-prediction is used to represent a CU and PU types determine which prediction type will be used to code a particular CU. Fig. 2 shows the processing order of 8×8 CUs in a 16×16 CU and the PU order within an 8×8 CU. For inter-prediction, PU types can be 2N×2N, 2N×N, N×2N or N×N where 2N×2N corresponds to the size of the CU. Motion vectors for inter-prediction are determined through motion estimation. If asymmetric motion partitions (AMP) are used, non-square PUs for inter-prediction also include 2N×nU, 2N×nD, nL×2N and nR×2N. It should be noted here that AMP partitions are not included in the hardware cost and coding efficiency analysis in this work but this analysis can be extended to cover these partition types as well. N×N is only used at the SCU level to avoid redundant representation. This is because N×N PU of a 16×16 CU can be represented with the 2N×2N PU at 8×8 CU level. This is true except for the SCU level so N×N is only used in an SCU.

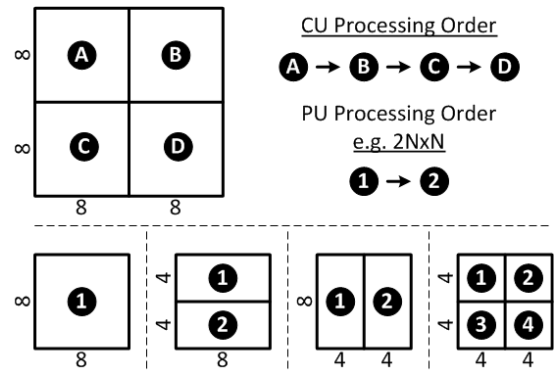


Fig. 2. Processing order for 8×8 CUs in a 16×16 CU is from A to D. For each 8×8 CU, PU types are also processed sequentially from 2N×2N to N×N. Finally inside a PU type, processing order is from 1 to 4.

B. *Motion Estimation in HEVC*

Motion estimation (ME) is one of the most critical blocks in video encoding in terms of implementation cost. Table II shows specifications of various recently published video encoders. It can be seen from Table II that ME accounts for a large fraction of total encoder area.

Motion estimation in HEVC is block-based where block sizes can be as large as 64×64 (LCU size) and as small as 4×4 (N×N PU in an 8×8 CU). A 64×64 LCU can be represented by various combinations of CUs and PUs. For example, 64×64 LCU can be represented by a single 2N×2N PU or it can be divided into 8×8 CUs where each CU is represented with four 4×4 blocks (N ×N PU type). In the former case, an LCU is represented with a single motion vector pair (one vector for horizontal displacement and one for vertical displacement) and in the latter case, with 256 pairs. For an LCU with many details, using smaller block sizes with separate vectors can provide better compression. In contrast, for large and smooth areas, using larger block sizes and fewer motion vectors can be more efficient. Hence, supporting all block sizes provides the highest flexibility and best coding efficiency but this also results in highest hardware implementation cost.

In hardware implementations, fast search algorithms [10-11] are widely used. These algorithms are extremely critical for the complexity as they determine the number of calculations and memory accesses which impact the area of hardware, its power consumption and lastly its memory bandwidth requirement. Moreover, the search algorithm's performance also affects the coding efficiency depending on how accurately this algorithm finds the motion.

Work	Standard	Resolution	Rate	Area	ME Area	Frequency	Power	Process
			fps	*mm²*	*%*	*MHz*	*mW*	*µm*
[6]	AVC/H.264	1280x720	30	31.7	80	108	785	0.18
[7]	MPEG4	640x480	30	7.7	55	28.5	18	0.18
[8]	AVC/H.264	1280x720	30	18.5	54	108	183	0.13
[9]	AVC/H.264	1920x1080	30	10.0	70	145	242	0.13

Table II. Comparison of previously published encoder chips. ME area is a significant portion of total chip area making ME a critical part of the encoder design.

Motion estimation is on the encoder side but a video compression standard only defines the decoder side. Hence, encoder side decisions can be different from one design to the other as long as the output of the encoder is compliant with the standard. The decisions made on the encoder side, however, affects coding efficiency.

In this paper, the encoder implementation given in the HM 3.0 software [12] using common conditions (single reference frame in each direction, fast motion search, no AMP and no merge mode) is used as the reference point. It should be noted that some of these modes (e.g. 4×4) is no longer supported in HEVC but this work uses HM-3.0 as a baseline implementation and results for the unsupported modes are kept in the analysis to provide a reference to the readers.

This paper presents motion estimation design considerations for HEVC standard with a focus on hardware implementation trade-offs. For hardware cost, we considered on-chip hardware and memory area as well as on-chip and off-chip bandwidth. The rest of the paper is structured as follows. Section II presents a hardware cost analysis for HEVC motion estimation and investigates the hardware cost vs. coding efficiency trade-off. Based on the results from Section II, Section III focuses on one of the possible motion estimation architecture configurations and this section talks about hardware-aware fast search algorithm development. Furthermore, hardware implementation details are presented in Section III. Lastly, Section IV concludes the paper.

II. Hardware Cost Analysis for HEVC Motion Estimation

HEVC reference software implementation (HM) is completely sequential on the processing of the CUs and PUs and consequently achieves highest coding efficiency. This is mainly due to the dependency of one block's cost calculations on neighboring blocks' motion information. Specifically, advanced motion vector prediction (AMVP) calculation requires the spatial as well as temporal neighbor information to create a list of motion vector predictors. This list is used to predict the motion vectors during motion search and signal the final motion vectors once the motion search is concluded [13].

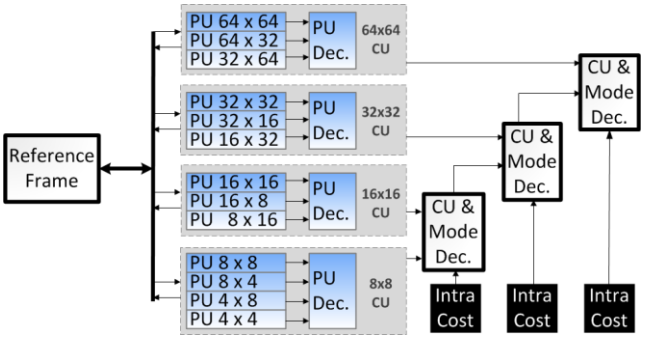


Fig. 3. Architecture for an HEVC motion estimation engine supporting all block sizes from 64×64 to 4×4 (except AMP partitions). "PU Dec." refers to PU decision. This architecture allows sequential processing of smaller blocks and can use exact motion information from neighboring blocks.

Hence, it is important to consider an architecture which is capable of implementing this sequential processing so we can quantify the hardware cost of realizing a motion estimation engine providing a coding efficiency that is equivalent to reference software.

Previous work [14-17] has discussed various simplifications to allow search range and cost calculations across various blocks to be shared in hardware. However, these simplifications cause motion vector predictions to be inaccurate and hence a degradation in coding efficiency.

This section presents an architecture that is capable of processing CUs and PUs sequentially and performing motion searches independently. Then, the hardware cost of HM's fast search algorithm is quantified with a methodology to estimate area and bandwidth. Finally, a trade-off analysis is done that compares different motion estimation configurations supporting only a subset of all CU sizes and PU types in terms of area, bandwidth and coding efficiency.

A. HEVC Motion Estimation Architecture

In hardware, HM's sequential processing of CUs and PUs requires separate and independent engines performing motion search for different block sizes. Block sizes are determined by the corresponding CU sizes and PU types. Fig. 3 shows an HEVC motion estimation engine architecture supporting all block sizes from 64x64 down to 4x4 except AMP partitions. This architecture can be generalized to cover AMP partitions as well. This architecture is designed to support real-time video encoding with the specifications shown in Table III.

There are a total of 13 engines in the architecture in Fig. 3: Three engines for each PU size (e.g. 32×32, 32×16 and 16×32 for the 32×32 CU) except for the 8×8 CU where there is a fourth engine to support N×N (4×4) partition. Each engine consists of blocks to perform AMVP list, integer motion estimation (IME), fractional motion estimation (FME) and a reference pixel buffer.

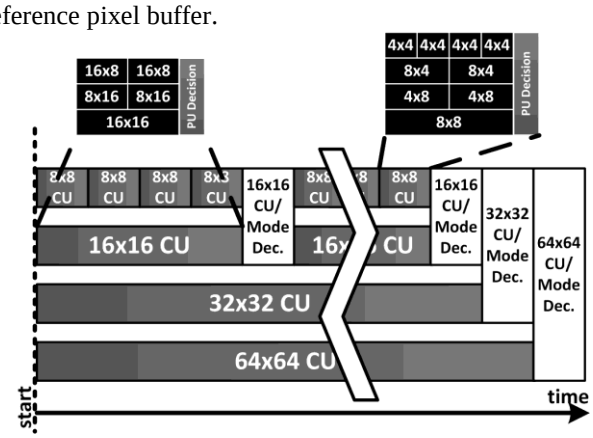


Fig. 4. Processing order of CUs and PU types inside CUs for the architecture in Fig. 3. For a 64×64 LCU, costs for smaller blocks are combined and then compared to larger block sizes to find the best combination of blocks providing the smallest cost for the entire 64×64 LCU.

The processing order for one 64×64 LCU is shown in Fig. 4. Motion searches are performed for four 4×4 blocks, two

8×4 and 4×8 blocks and one 8×8 block. Then a PU decision is made to decide which PU type provides the smallest cost for the first 8×8 CU. Similarly, three more 8×8 CUs are processed sequentially and their costs are output to CU & Mode Decision block. During this time, PU decision for the first 16×16 CU is also finished and a decision can be done for the first 16×16 CU. This continues until an entire 64×64 LCU is processed by all engines. It should be noted that intra/inter decision is done at the CU level and hence costs associated with intra prediction are being provided as external inputs to make an intra/inter decision. It is also important to note that, for a fixed throughput constraint, cycle budget to process a smaller block size is tighter. Hence, data bandwidth requirements can be significantly larger for smaller block sizes compared to larger block sizes. Consequently, smaller block sizes impose a larger hardware cost.

B. Overview of Hardware Cost Analysis

The following part of this paper will be talking about the hardware cost analysis of HEVC motion estimation module. The top level architecture given in the previous section will be used for this analysis and the algorithms used in HM-3.0 implementation will be analyzed.

Specifications of an HEVC Encoder	
Maximum Resolution	4K×2K i.e. 3840×2160
Maximum Frame Rate at 4K×2K	30
LCU Size	64×64
# of Ref. Frames	1 in each direction
Search Range	64 in x- and y-dir.
Frequency of Operation	200MHz
Process Technology	65nm Low-Power CMOS

Table III. Specifications for an HEVC encoder considered in this work. The design can support real-time encoding at 4K×2K at 30fps with a clock frequency of 200MHz.

The specifications of a target encoder are given in Table III but this analysis can be extended for other encoder implementations. For hardware cost, we will consider logic and on-chip memory area as well as on- and off-chip data bandwidth requirements.

Logic Area Estimation Method and Results

For logic area estimation, the methodology used is as follows:

1. Implement basic building blocks in hardware and use synthesis tools to get unit area and power numbers at the target frequency of operation point.
2. Calculate the amount of parallelism required for throughput constraint.
3. Estimate total area by using unit numbers and amount of parallelism.

In the top level architecture given in Fig. 3, there are a total of 13 parallel engines. Looking at the number of pixel calculations/cycle, they are found to be constant across parallel engines. Although the number of available cycles is getting larger from smaller blocks to larger blocks, number of computations/block is also getting larger with the same factor. Hence, the hardware required for different engines to perform search candidate evaluation is mostly constant.

Total area of one engine including IME, FME and AMVP blocks is estimated to be 305k gates in a 65nm CMOS process. It is important to note that the entire motion estimation module in Fig. 3 consists of 13 engines, resulting in roughly 4M gates. Moreover, to support forward and backward motion estimation of the random-access configuration, this number needs to be scaled up by roughly a factor of two.

On-Chip Memory Size Estimation Method and Results

As explained in Section 2, each motion estimation engine in Fig. 3 is performing independent searches and for each engine, a separate memory is necessary in each direction (forward and backward) and for each reference frame. Table IV shows the size of on-chip memory needed to support ±64 search range. Extra pixels are necessary for pixel interpolation in fractional motion estimation and they are included in calculations.

Block Size	On-Chip Mem. Size	Block Size	On-Chip Mem. Size
64×64	39kB	16×8	21kB
64×32	33kB	8×16	21kB
32×64	33kB	8×8	20kB
32×32	28kB	8×4	20kB
32×16	25kB	4×8	20kB
16×32	23kB	4×4	19kB
16×16	23kB		

Table IV. On-chip reference buffer size needed for each engine to support ±64 search range for a single reference frame.

A total of 0.65MB of on-chip memories is necessary to support a single reference frame in forward and backward directions for the entire motion estimation module in Fig. 3. This number heavily depends on the selected search range size. The search range size can be reduced at the expense of coding efficiency loss. The work in [18] quantifies this effect and reports up to 3.5% loss in coding efficiency when search range is reduced from ±64 to ±16. For frame resolutions up to 4K×2K, a larger search range is advantageous and this work uses ±64 in both directions for this analysis.

It should be noted that on-chip memory size for small block sizes is not significantly lower than the size for larger block sizes (39kB for 64x64 and 19kB for 4x4). Consequently, smaller block sizes do not provide a significant advantage in terms of memory size.

Additional on-chip storage (e.g. line buffers for motion information) can be necessary for AMVP but the size heavily depends on the specific implementation and the target resolution. Moreover, these buffers can be shared across parallel engines. For this work, on-chip line buffers are

considered for motion information of the top line in forward and backward directions. For a 4K×2K video encoder, the amount of storage is estimated to be around 30kB.

On-Chip and Off-Chip Bandwidth Estimation Method and Results

On-chip and off-chip bandwidth are critical in hardware implementations as these numbers affect system power consumption and can be limiting factors.

On-chip bandwidth is determined by the size of reference buffer for each engine and how frequently it is accessed. For the fast search algorithm in HM, during IME, entire search range can be accessed. This occurs in the case of complex motion. To capture the worst-case upper limit, it can be assumed that the entire search range in the reference buffer is accessed for every block. On-chip bandwidth for FME is significantly smaller as only a refinement is done at this stage. Lastly, bandwidth for motion information of neighboring blocks that is necessary for *AMVP* candidate calculations is small compared to the on-chip bandwidth of the integer and fractional motion estimation.

The reference frames for high-definition sequences are often too large to store on-chip thus they are stored on an off-chip memory and the necessary parts of these reference frames are transferred to on-chip buffers before processing. Off-chip bandwidth considered here is the off-chip memory's read bandwidth to bring reference pixel data from off-chip to the on-chip buffers for motion estimation. Similarly, off-chip bandwidth is determined by the size of the reference buffer and how frequently reference buffers for each engine need to be updated. Because of the correlation of motion between neighboring blocks, in the ideal case, data re-use between consecutive blocks can be close to 100%. However, it should be noted that the processing order of CUs and PUs in an LCU (Fig. 4) does not allow 100% data re-use and hence causes the same part of the reference window to be read multiple times. Increasing size of the on-chip buffer can improve the data re-use at the expense of larger on-chip memory area. In this section, minimum buffer sizes given in the previous sub-section (Table IV) are assumed in the bandwidth calculations.

Table V shows on- and off-chip bandwidth requirement for each engine. It should be noted that small block sizes such as 4x4 require a very large on-chip and off-chip bandwidth compared to larger block sizes and imposes a higher cost for hardware implementation.

Block Size	On-Chip BW	Off-Chip BW	Block Size	On-Chip BW	Off-Chip BW
64×64	2.2	1.49	16×8	39.6	13.72
64×32	3.8	1.86	8×16	39.6	10.33
32×64	3.8	1.48	8×8	75.6	17.47
32×32	6.4	3.64	8×4	145.9	30.21
32×16	11.5	6.05	4×8	145.9	22.94
16×32	11.5	5.20	4×4	283.8	36.92
16×16	20.9	7.62			

Table V. On- and off-chip bandwidth requirements for each engine in Fig. 3 with a search range of ±64. All numbers are in GB/s. On-chip bandwidth numbers reflect the worst-case condition and off-chip bandwidth numbers assume maximum data reuse between consecutive blocks.

Hardware Cost vs. Coding Efficiency Trade-Offs

In this section of the paper, we will analyze various motion estimation configurations where some block sizes (i.e. CU sizes and PU types) are not supported and consequently we need less than 13 engines. However, the coding efficiency will be worse because of the exclusion of some block sizes. It is important to quantify the savings in hardware and loss in coding efficiency to be able to make an optimum decision between supported block sizes.

Fig. 5 shows hardware area and bandwidth as well as coding efficiency results for 11 different motion estimation configurations. Each column corresponds to a different configuration supporting all or some of the available block sizes.

	Configuration #										
	1	2	3	4	5	6	7	8	9	10	11
64x64	Y	Y	Y	Y	Y	Y	Y	N	N	N	N
64x32	Y	Y	N	Y	N	Y	N	N	N	N	N
32x64	Y	Y	N	Y	N	Y	N	N	N	N	N
32x32	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N
32x16	Y	Y	N	Y	N	N	N	Y	N	N	N
16x32	Y	Y	N	Y	N	N	N	Y	N	N	N
16x16	Y	Y	Y	Y	Y	N	N	Y	Y	Y	Y
16x8	Y	Y	N	N	N	N	N	Y	N	Y	N
8x16	Y	Y	N	N	N	N	N	Y	N	Y	N
8x8	Y	Y	Y	N	N	N	N	Y	Y	Y	Y
8x4	Y	N	N	N	N	N	N	N	N	N	N
4x8	Y	N	N	N	N	N	N	N	N	N	N
4x4	Y	N	N	N	N	N	N	N	N	N	N
Logic Area (M gates)	7.9	6.1	2.44	4.27	1.83	2.44	1.22	4.27	1.83	2.44	1.22
Ref. Buffer Size (KB)	680	565	248	439	208	234	163	356	170	201	115
On-Chip BW (GB/s)	1581	429	209	121	59	32.5	17.3	409	205	351	192
Off-Chip BW (GB/s)	159	69	30.2	27.4	12.7	8.5	5.1	64	28.7	49.1	25.1
Bit-Rate Increase (%)	0	2	3	12	12	34	34	3	4	7	11

Fig. 5. Hardware cost vs. coding efficiency comparison for 11 different motion estimation configurations. "Y" and "N" represents if a block size is supported or not respectively.

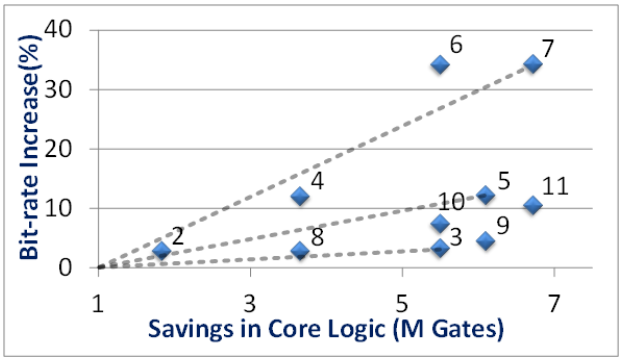
Configuration #1 supports all block sizes and is the anchor configuration for this work. Simulations in HM-3.0 are performed to quantify coding loss for each configuration except for the configurations #8-11 where HM-3.2 is used because of a bug in HM-3.0 which prevents LCU size to be changed. The bit-rate increase in Fig. 5 is given as the average of the numbers from all-intra, low-delay, low-delay P and random-access common test conditions defined by JCT-VC [4]. The common test conditions cover a wide range of sequences with resolutions as small as 416×240 and as large as 2560×1600.

Fig.6-a and Fig. 6-b plot core area savings vs. bit-rate increase and off- chip bandwidth savings vs. bit-rate increase for 10 configurations in Fig. 5 with respect to the anchor, configuration #1. Each configuration is denoted by a dot on this figure except for the anchor configuration as the anchor would be at the origin of the plot. The slope of the lines connecting each configuration to the origin provides a visual method to compare how efficient each configuration is. A smaller slope means that more savings can be achieved with smaller bit-rate increase (coding loss). Lines connecting

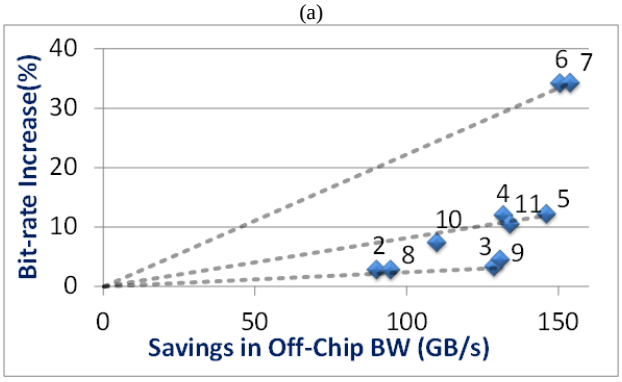
configurations #3, #5 and #7 and the origin are given on Fig. 6-a and Fig. 6-b as examples.

Observations and Conclusions

It can be observed from Fig. 5 and Fig. 6 that configurations supporting smaller block sizes such as 4×4 require largest area and bandwidth although the coding gain achieved through supporting them is relatively smaller. In other words, not supporting smaller partitions has a smaller effect on coding efficiency although these engines contribute significantly to bandwidth and area. For example, by removing 4x4, 4x8 and 8x4 block sizes in configuration #2, 17% memory area, 3.7× on-chip bandwidth and 2.3× off-chip bandwidth can be saved at the expense of only 2% coding loss with common conditions using a single reference frame and fast search algorithm. This result supports the decision about removing 4x4 PU and 4x8 and 8x4 bi-prediction from the final standard.



(a)



(b)

Fig. 6. (a) Core area savings vs. bit-rate increase and (b) off-chip bandwidth savings vs. bit-rate increase scatter plots for all the configurations given in Fig. 5.

Another observation from Fig. 5 and Fig. 6 is that not supporting 2N×N and N×2N does not result into significant coding efficiency loss. Fox example, from configuration #2 to #3, coding efficiency degrades by 1% and the degradation from configuration #4 to #5 and #6 to #7 are less than 1%.

On-chip reference buffer size mainly depends on the search range and block size. However, from smaller to larger block sizes, the increase in memory size is not very significant. In terms of memory bandwidth, small block sizes, especially smaller than 8x8, impose very high bandwidth requirements. If savings are necessary due to system level restrictions for bandwidth, small block sizes can be chosen not to be supported.

Lastly, final decision on supported block sizes depends on the area and bandwidth limitations as well as coding efficiency specifications of the target encoder. Since larger area and higher bandwidth often result in higher power consumption, battery- powered mobile applications might trade-off some of the coding efficiency for lower power consumption. If coding efficiency has the highest priority, all block sizes can be supported (configuration #1) although this might lead to a significantly large area and power consumption. If area as well as power are critical, configuration #5 and #7 are suitable solutions.

III. Cost and Coding Efficient (CCE) HEVC Motion Estimation Design

In this section we will talk about architecture and algorithm development for reducing the hardware cost even further with minimum impact on the coding efficiency. It should be noted that although the following algorithm and architecture developments are targeted for configuration #5, these algorithms and architectures are suitable for all configurations supporting square-shaped block sizes. Moreover, the hardware implementation details and results are also provided based on the proposed algorithms presented in this section and the target specifications given in Table III.

A. *CCE Motion Estimation Architecture*

Top level architecture for CCE motion estimation module is given in Fig. 7. CU sizes of 64×64, 32×32 and 16×16 are supported. Since there is only a single PU type (2N×2N) in each CU engine, an internal PU decision is not necessary. It should be noted that this architecture is still capable of processing blocks sequentially and consequently using exact motion information of the neighboring blocks.

B. *Search Algorithm Development for CCE Motion Estimation*

Fast search strategy used in HM-3.0 starts the search around the best AMVP and consists of many inter-dependent stages. For example, the result of the initial diamond search determines if a sub-sampled raster search is performed or not. In hardware implementation, this dependency increases complexity and often results in extra cycles or extra hardware to account for the worst-case conditions.

Recent work focused on search algorithms that can be parallelized in hardware implementation [8, 16]. For CCE implementation, we implemented a similar, two- stage search strategy for IME where each stage can be independently performed in parallel. Fig. 8 shows IME search patterns used in each of the stages. First, search center is decided by comparing AMVP list entries (up to three entries) and [0,0]. During this comparison, SAD (sum of absolute differences) cost is used. After search center is determined, two stages of the search is started in parallel.

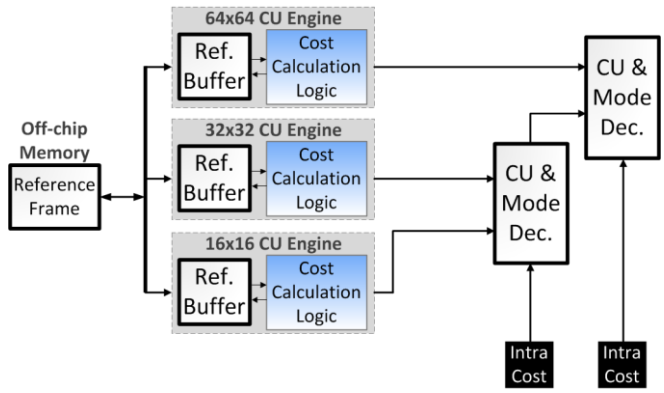


Fig. 7. Top level architecture of the CCE motion estimation implementation. Block sizes of 64×64, 32×32 and 16×16 are supported.

One of the parallel stages consists of a coarse search covering ±64 by checking every 8th candidate in each direction. This stage can capture a change in motion or irregular motion patterns that cannot be tracked by AMVP. The second stage performs a more localized three step search around the ±7 window of the search center. This stage can capture regular motion. There are two additional advantages of running both searches in parallel. First, the pixel data for the coarse search can be used to perform the localized three step search hence reducing memory bandwidth. Secondly, the cycles necessary to access the pixels for both search stages can be shared to reduce the total number of cycles.

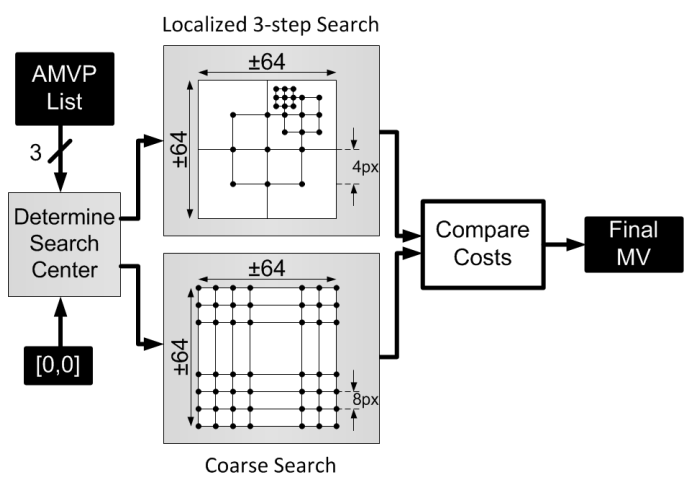


Fig. 8. Two stage search approach used for CCE implementation. Stages are independent of each other and can be performed in parallel in hardware.

It is important to note that the AMVP calculation for all CUs uses exact motion vectors of the neighbors and AMVP is accurate and hence can track motion well in most cases.

The proposed IME search strategy checks a total of 285 candidates for each CU as opposed to up to 848 candidates that are checked in fast search strategy in HM-3.0. This results in roughly 2× hardware area reduction in IME for the same throughput constraint. Actual savings might be larger in implementation because of the additional complexity due to inter-dependent stages of HM-3.0 algorithm. Lastly, for FME, search strategy of HM is used where sub-pixel positions are evaluated around the best integer motion vector to find the best sub-pixel accurate motion vector.

C. Sharing Reference Pixel Buffers for CCE Motion Estimation

Sharing the on-chip reference buffer across parallel engines can provide significant savings in terms of area and off-chip bandwidth. However, restricting the search range of parallel engines to a shared window will result in coding efficiency loss. This loss can be minimized by determining the shared search window carefully.

In the case of separate reference buffers with ±64 search range for each engine, the implementation in Fig. 7 requires three 1R1W (1 read, 1 write) port memories with 39kB, 27.5kB and 22.5kB sizes for 64×64, 32×32 and 16×16 engines respectively as given in Table IV. Total area consumed by these three memories can be estimated to be roughly 1.25mm^2 in a 65nm CMOS technology [19] as shown in Table VI. It should be noted that this area is for storing the pixels on the chip for a single direction and single reference frame.

In contrast to this, in the case of a shared reference buffer with ±64 search range, the size is determined by the largest CU size and a single 39kB memory is needed with 3R1W ports. Although the bit-cell area and some peripheral components need to be expanded to support multiple read ports, the overall area can be smaller as shown in Table VI. Hence, shared search range across parallel engines results in 16% area savings for the implementation considered in Fig. 7.

	Separate Buffer (3 x 1R1W)	**Shared Buffer (1 x 3R1W)**
Memory Size	89kB	39kB
Est. Cell Area	0.85μm^2	1.55μm^2
Est. Array Area	0.75mm^2	0.61mm^2
Est. Periphery Area	0.5mm^2	0.44mm^2
Est. Total Area	1.25mm^2	1.05mm^2

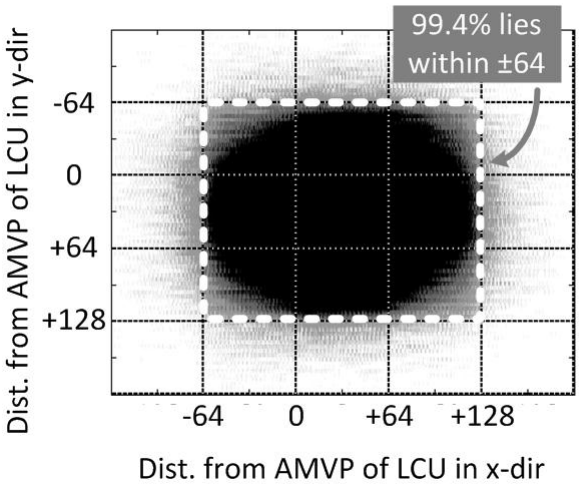
Table VI. Area comparison of shared and separate reference buffers. Estimates are based on a 65nm CMOS technology.

With independent motion searches, each engine might have different search centers and consequently access different parts of the reference frame as the search window. Table VII shows maximum and average off-chip bandwidth for 64×64, 32×32 and 16×16 engines. The upper limit on the bandwidth is calculated by assuming that the entire on-chip reference buffer needs to be updated between consecutive CUs and hence no data re-use is possible. The total maximum off-chip bandwidth is 29.5 GB/s for supporting 4K×2K resolution at 30fps assuming a search range of ±64. Average bandwidth number with close to 100% data re-use between consecutive LCUs is 12.7 GB/s. In the case of a shared reference window across engines, the maximum bandwidth is equal to the maximum bandwidth of the 64×64 LCU since the size of the shared search window is determined by the largest CU size given that the reference pixel data for smaller CUs are part of the data for the LCU. Sharing the search window provides 13.4× and 8.3×

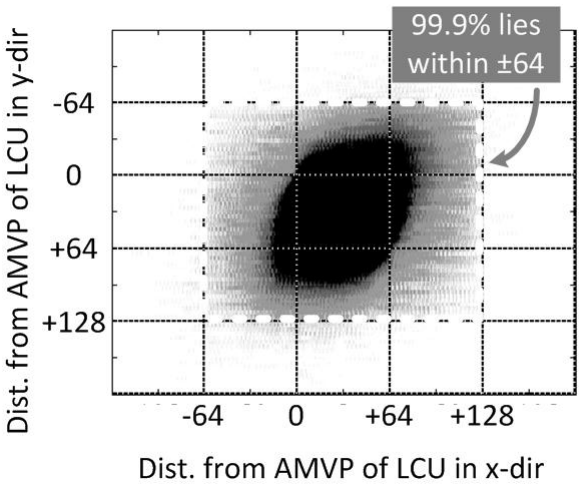
savings in terms of the maximum and average bandwidth requirements.

CU Size	Max. Off-Chip BW	Avg. Off-Chip BW
64x64	2.2GB/s	1.5GB/s
32x32	6.4GB/s	3.6GB/s
16x16	20.9GB/s	7.6GB/s
Total	29.5GB/s	12.7GB/s

Table VII. Maximum and average off-chip bandwidth requirement for different CU sizes (search range is ±64) for supporting 4K×2K at 30fps. Average off-chip bandwidth is calculated by an experiment on Traffic sequence under random access condition.



(a)



(b)

Fig. 9. (a) Density maps for the relative location of pixels from best-matching blocks with respect to the AMVP of the LCU for (a) PeopleOnStreet and (b) Traffic sequences. More than 99% of the pixels lie within ±64 (a 192x192 block of pixels surrounding a 64x64 CU) of the AMVP of the LCU (2560x1600 sequences with QP=22 in random-access configuration).

In order to minimize the coding efficiency impact of sharing search window across engines, a good representative should be selected for the motion of all CUs within an LCU. AMVP of the LCU is observed to provide a good center point for the shared search window. Fig. 9 shows the density map for the relative location of the pixels from best matching blocks with respect to the AMVP of the LCU for two different sequences. Best matching blocks are calculated with the original HM-3.0 fast search algorithm and the search range is ±64 pixels in each direction. For both sequences, more than 99% of the best matching pixels lie in the ±64 vicinity of the AMVP (192x192 block of pixels surrounding the 64x64 block that AMVP is pointing to) of the LCU. This indicates that AMVP of the LCU can be used as the search window center without introducing significant coding efficiency loss.

For smaller CUs that have different AMVPs and consequently different search centers, the search window is modified to fit in the shared window such that the window is shifted to make sure it lands inside the shared search window in the on-chip buffer. It is important to note that although the search window is modified, original AMVP of the CU is used in cost calculations. Moreover, total number of candidates stays the same for all CU sizes regardless of the search window being modified or not. This provides simplicity in hardware implementation.

D. Reference Pixel Data Pre-fetching Strategy

For a practical hardware implementation, off-chip memories are used for large storage requirement of reference frames. DRAMs are generally used to implement these off-chip storage. Because of the internal mechanism of DRAMs, it is necessary to request the data from off-chip memories in advance since the latency of these memories can be on the order of thousands of cycles. Stalling the encoding operation while waiting for the pixel data from DRAM can cause a reduction of the throughput of the system.

To address this, the pre-fetching strategy described in [20] is implemented for CCE motion estimation. This strategy involves calculating the center of the reference window by only using the information from the top row such that the requests for the pixels can be placed in advance.

E. Enlarging On-Chip Reference Buffers for Higher Data Reuse Rate

In order to share the cycles between writing to and reading from the reference buffer, larger on-chip storage is necessary. This extra storage is used to start writing the data for the next LCU while motion estimation for current LCU is continuing. For this purpose, an extra storage that is 64 pixels wide (size of an LCU) is necessary. Obviously, extra storage alone is not adequate if the search center from current LCU to next LCU is changing. This issue can be addressed by allowing a larger storage for reference buffers and algorithm modifications.

In the ideal case where consecutive LCUs have the same AMVP, a 100% data reuse rate can be achieved where search window moves to the right by 64 pixels for every LCU. An illustration of 100% data reuse case is shown in Fig. 10-a, where five LCUs and their corresponding search window are shown. However, this is highly unlikely and AMVP of consecutive LCUs can be very different from each other especially in frames with complex motion. Fig. 10-b shows a case where data reuse between five LCUs is very poor.

In the discussion above, we always considered the case where on-chip reference buffer size is equal to the search window size and additional storage for the next LCU. However, if the on-chip memory size is increased to hold a larger window, data reuse rate can be improved as there is a higher chance of the data on the chip matching next LCU's search window. Although larger on-chip memories result in larger bandwidth per LCU, the improvement in data reuse rate can over-power this increase and results in a reduction in overall average bandwidth.

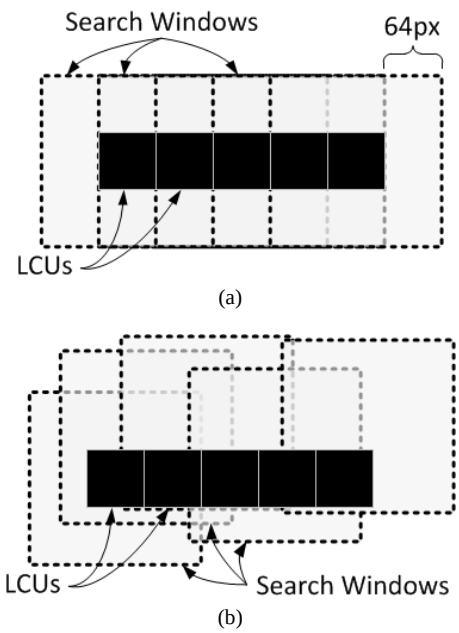


Fig. 10. Search ranges of five consecutive LCUs with (a) uniform motion maximizing data reuse and (b) non-uniform motion causing lower data reuse rate.

It should be noted that although on-chip memories hold a larger window, search window is not increased and kept as ±64 in each direction and consequently the total number of candidates in motion search is not affected from this modification. Fig. 11 shows the reference window with N extra pixels on each side and also the extra 64 pixels for the next LCU.

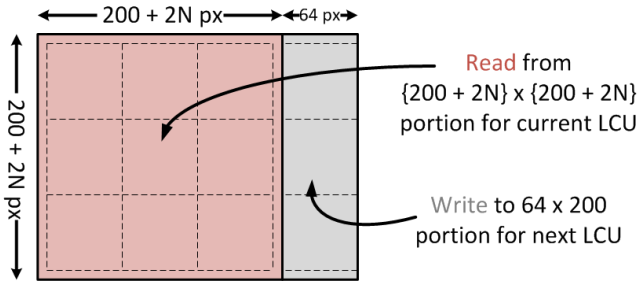


Fig. 11. Extra storage is needed for on-chip buffers to share cycles for read and write accesses to the memories and N extra pixels on each side of the reference buffer is considered for improving data reuse rate. {200+2N}×{200+2N} portion is used for current LCU and 64×{200+2N} portion is used for next LCU. LCU size is 64×64 and search range is ±64.

The effect of increasing reference buffer size by N pixels on all four sides is analyzed in terms of bandwidth. Fig. 12 plots total off-chip bandwidth, maximum data reuse rate and on-chip buffer size for two different sequences with changing N.

With increasing N, on-chip buffer size and the bandwidth due to updating a larger buffer for every LCU increase. However, also with increasing N, maximum data reuse rate increases. Fig. 12 also shows the bandwidth with 0% data reuse without any increase in on-chip buffer size (i.e. N=0) and the bandwidth with 100% data reuse with N=32.

Because of the conflicting trends, write bandwidth makes a minimum around N = 16 for both sequences. This provides close to 1.8X savings in off-chip bandwidth at the expense of 35% area increase in reference pixel buffers.

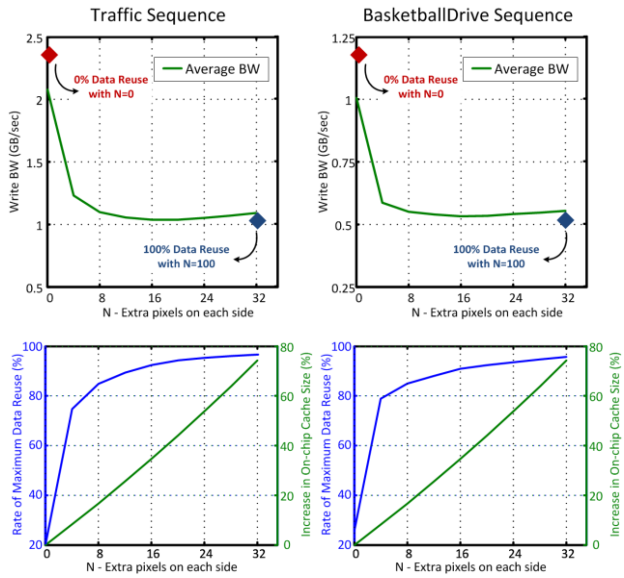


Fig. 12. Total off-chip write bandwidth, maximum data reuse rate and on-chip buffer size for Traffic (2560×1600) and BasketballDrive (1920x1080) sequences. Simulations are performed in Random Access test condition with QP = 22.

To further improve data reuse rate and reduce off-chip bandwidth, pre-fetching algorithm is modified to limit the difference between two AMVPs (centers of search windows) to ±N. Intuitively, this translates to the search window being able to track changes in motion by at most N pixel step sizes. For this work, N is chosen to be 16 to minimize its effect on the coding efficiency and to minimize total bandwidth.

F. Effect on Bit-Rate

The changes in various parts of the search strategy for CCE motion estimation are implemented in the HM-3.0 software and their effect on coding efficiency is quantified under common conditions. Simulations are performed under the conditions defined in [4].

Table VIII shows coding efficiency change with respect to the HM-3.0 fast search algorithm in configuration #5 after each modification. Columns LD, LDP and RA stands for low-delay, low-delay with P and random-access test conditions as

defined by JCT-VC [4]. *Avg* column is the average of LD, LDP and RA. Lastly, *Max* and *Min* columns are the maximum and minimum rate change for all tested sequences respectively.

The average cumulative rate increase due to the proposed changes is 1.6%. Search algorithm changes constitute 1% of this increase. Random-access test conditions result in the largest coding efficiency degradation as the distance between the reference frame and the coded frame is longer in this test condition. In general, sequences with lower resolution face a larger degradation of coding efficiency compared to the sequences with higher resolution because of the architectural decision to not support CU sizes smaller than 16x16.

G. Hardware Implementation of a CU Engine

Hardware implementation results presented in this section are based on the CCE ME implementation discussed in Section III.

Fig. 13 shows the architecture of one CU engine. Integer and fractional motion estimation parts are implemented together and they are not pipelined for maximum coding efficiency as pipelining these processes would require integer motion vectors to be used in the AMVP calculations.

Reference buffer and block buffer hold reference and current CU's data respectively. Reference buffer write control exerts write operations on the reference buffer for the next LCU whereas read control accesses the search range data. AMVP part calculates the motion vector predictor list. Cost tree and comparator array are capable of calculating the cost of 4 motion vector candidates/cycle for the 16×16 CU for which the cycle budget is the shortest. For larger CU sizes, although the number of cost calculations is the same, the throughput is lower (e.g. 1 motion vector candidate/cycle for 32x32 CU).

Best position and cost is stored in sequential elements and compared against costs for newer candidates. Finally, engine control ensures the flow of data inside the engine as well as the communication of higher level control units.

This design is targeted towards an encoder supporting real-time processing of 4Kx2K frame resolution at 30fps with a 200MHz clock as given in Table III. These specs require the processing of each 64x64 LCU to be completed in 3292 cycles and the hardware design is parallelized to provide this throughput.

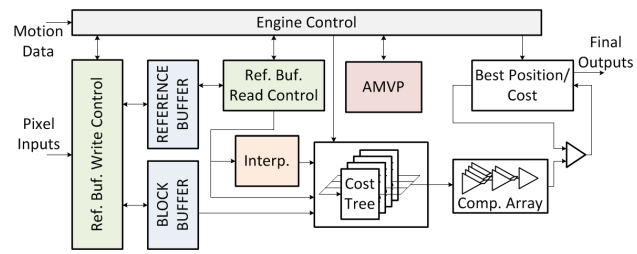


Fig. 13. Architecture of one engine in CCE HEVC motion estimation implementation.

To be able to support the 4 motion vector candidates/cycle output requirement for the 16x16 CU engine, search range is partitioned into 88 blocks of SRAMs, each block holding roughly 200 words and four neighboring pixels on every word.

Fig. 14 shows the allocation of pixels on memory banks. Going from one LCU to the next, since most of the data is reused, only pointers to the memory locations are changed. This is handled in the read control by holding the left-top coordinate (Left-TopX, Left-TopY) of the search range as well as an address bias (AB) which is incremented by 64 pixels for every LCU. Engine control requests a stripe (8×44) of reference pixels by providing the left-top coordinate (InX,InY) to read control. After data is read from SRAM blocks, 8×44 pixel block is output in the next cycle. There is a multiplexer

	# of Search Candidates	**On-Chip Buffer Size (mm^2)**	**Off-Chip BW (GB/s)**	**LD**	**LDP**	**RA**	**Avg**	**Max**	**Min**
HM-3.0 Anchor (Conf. #5)	466	1.25	12.7	0	0	0	0	0	0
Search Algorithm	285	1.25	12.7	0.6	0.8	1.6	1.0	3.1	0.1
Search Algorithm & Shared Search Range	285	1.05	1.5	0.6	1.0	2.9	1.5	7.4	0.2
Search Algorithm & Shared Search Range & Pre-fetch	285	1.05	1.5	0.9	1.0	2.9	1.6	7.3	0.2
Search Algorithm & Shared Search Range & Pre-fetch & Limited Search Range Movement with N=16	285	1.38	1.25	0.9	1.0	2.9	1.6	7.4	0.2

Table VIII. Simulation results for the coding efficiency change after the search algorithm, shared search window, pre-fetching and limiting the movement of search range center by N = 16 with respect to HM-3.0 (configuration #5). Number of search candidates, on-chip buffer size and off-chip bandwidth numbers are also provided for comparison. All columns with coding efficiency change (i.e. LD, LDP, RA, Avg, Max and Min) are in percentage values.

array at the output of the read control to select appropriate outputs from SRAM blocks and put them in order.

New data overwrites the older data sequentially for every LCU in the reference buffer. At the beginning of an LCU line in the frame, all memory locations need to be updated. For all other LCUs, a 64×232 block (as explained in Fig. 11 with N=16) and possibly N=16 pixel wide edges are updated since, at the algorithm level, the movement of the search center is limited to be less than N=16 pixels between consecutive LCUs. Lastly, the search range accessed by the read control and the pixels that are overwritten by write control are not overlapping so read and write operations can be done in the same cycle.

Synthesis results for the reference buffer read and write control show that a total of 52.6k gates are used. Read control takes up a larger area due to the multiplexers to select the outputs from 88 SRAM blocks.

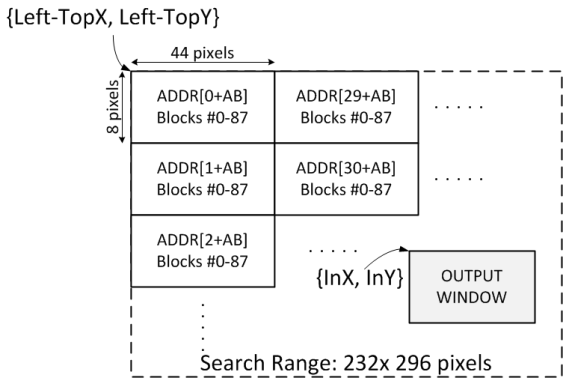


Fig. 14. Search range partitioning and physical location of pixels in memory banks for the search range shown in Fig. 11 with N=16.

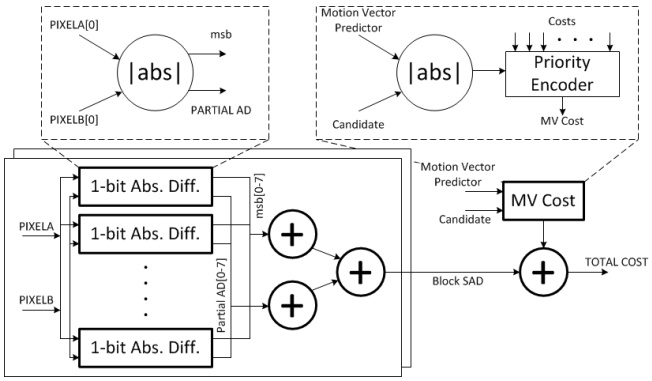


Fig. 15 Cost tree implementation using 1-bit absolute difference (AD) and motion vector cost calculation.

As shown in Fig. 15, cost tree calculates costs and adds the motion vector cost to create total motion cost. 1-bit partial absolute-differences (AD) are calculated and 1-bit 'msb' information is propagated to the output to make the critical path shorter. ADs and msb bits from multiple pixels are summed in parallel. MV cost calculation is implemented with a priority encoder as shown in Fig. 15. The input to the priority encoder is the absolute difference of the candidate and the motion vector predictor. Then, comparator array compares costs of candidates with the smallest cost and decides if the smallest cost needs to be updated or not. At the end of the search, smallest cost and its corresponding candidates are signaled as motion vectors. Lastly, cost tree and comparator array implementation results in 131k gates.

Fig. 16 shows the implementation of the AMVP block. A0-1 and B0-2 are spatial neighbors and C and H are temporal neighbors to the current block [13]. A scaling operation is used if the motion information from the neighbors cannot be used directly. The scaling operation involves two multiplication operations and constitutes a large fraction of the overall area of the AMVP block. Micro architecture of the scaling block is shown in Fig. 17. Picture order count (POC) values of the current and reference frame as well as the POC values of the neighboring blocks are used to calculate the scaling factor. Two multiplication operations are pipelined to meet the frequency requirements. Once the motion vector predictor candidates are calculated, they undergo a "uniquify" operation to ensure that the final AMVP list is composed of distinct members. AMVP block results in 26k gates.

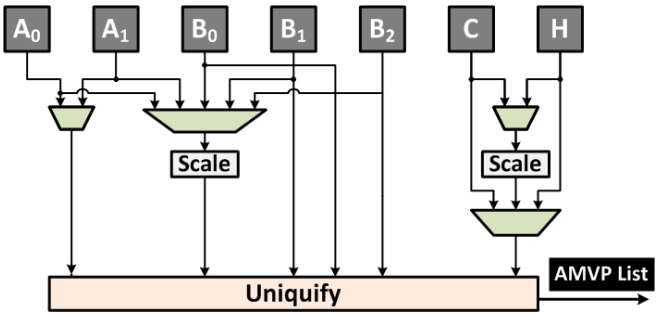


Fig. 16. Block diagram of the AMVP block.

IV. Conclusion

Motion estimation is one of the most critical blocks in HEVC encoder designs, and is analyzed for its hardware implementation cost in this work. This study presents the trade-offs between coding efficiency and hardware cost in order to make critical design decisions. Specifically, a motion estimation implementation providing coding efficiency equivalent to the reference software is considered and its hardware cost is quantified. This design is found to be very costly in hardware.

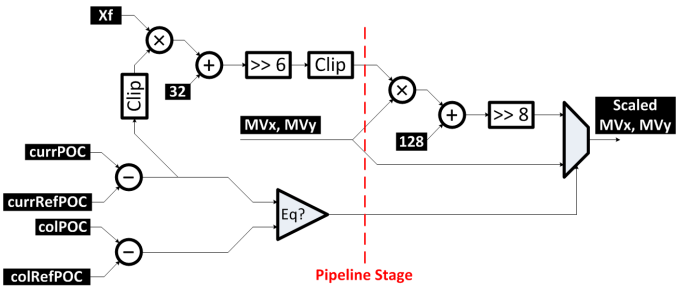


Fig. 17. Implementation of the scaling unit in AMVP.

To reduce hardware cost, first, a reduction in the number of coding engines is considered and quantitative analysis has

been performed to find the configuration providing the best trade-off. Secondly, to further reduce hardware cost, hardware-oriented algorithms are developed that are suitable for the selected architecture. Overall, 56x on-chip bandwidth, 151x off-chip bandwidth, 4.3x core area and 4.5x on-chip memory area savings are achieved when compared to the hardware implementation of the HM reference software design. These savings are achieved at the expense of <4% coding efficiency degradation with respect to the HM-3.0 supporting all CU sizes and PU types and with fast search. Finally, the methodology used in this work can be generalized to other parts of a video codec design for understanding hardware cost and coding efficiency trade-offs and eventually to make critical design decisions.

REFERENCES

[1] "Cisco Virtual Networking Index: Global Mobile DataTraffic Forecast Update, 2011-2016" [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white paper c11-520862 .html.

[2] J. Ostermann, P. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H.264/AVC: Tools, Performance and Complexity," *IEEE Circuits and Systems Magazine*, vol. 4, pp. 7–28, 2004.

[3] G. Delagi, "Harnessing technology to advance the next-generation mobile user-experience," *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 18 –24, Feb. 2010.

[4] "Joint Call for Proposals on Video Compression Technology," *ITU-TSG16/Q6, 39th VCEG Meeting: Kyoto*, 17-22 Jan. 2010, Doc. VCEGAM91.

[5] J. Ohm, G.J. Sullivan, H. Schwarz, Tan Thiow Keng, T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC),"Circuits and Systems for Video Technology, IEEE Transactions on , vol.22, no.12, pp.1669,1684, Dec. 2012.

[6] Y.-W. Huang, T.-C. Chen, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, C.-S. Chen, C.- F. Shen, S.-Y. Ma, T.-C. Wang, B.-Y. Hsieh, H.-C. Fang, and L.-G. Chen, "A1.3TOPS H.264/AVC single-chip encoder for HDTV applications," *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 128–129, Feb. 2005.

[7] C.-P. Lin, P.-C. Tseng, Y.-T. Chiu, S.-S. Lin, C.-C. Cheng, H.-C. Fang, W.-M. Chao, and L.-G. Chen, "A 5mW MPEG4 SP encoder with 2D bandwidth-sharing motion estimation for mobile applications," *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 1626–1627, Feb. 2006.

[8] H.-C. Chang, J.-W. Chen, C.-L. Su, Y.-C. Yang, Y. Li, C.-H. Chang, Z.-M. Chen, W.-S. Yang, C.-C. Lin, C.-W. Chen, J.-S. Wang, and J.-I. Quo, "A 7mW-to-183mW Dynamic Quality-Scalable H.264 Video Encoder Chip," *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 280–281, Feb. 2007.

[9] Y.-K. Lin, D.-W. Li, C.-C. Lin, T.-Y. Kuo, S.-J. Wu, W.-C. Tai, W.-C. Chang, and T.-S. Chang, "A 242mW 10mm2 1080p H.264/AVC High-Profile Encoder Chip," *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 314–315, Feb. 2008.

[10] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *International Conference on Information, Communications and Signal Processing (ICICS)*, pp. 292 –296, Sep. 1997.

[11] T.-H. Tsai and Y.-N. Pan, "A novel predict hexagon search algorithm for fast block motion estimation on H.264 video coding," *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 609 – 612, Dec. 2004.

[12] "JCT-VC Reference Software HM-3.0," *ISO/IEO MPEG and ITU-T*.

[13] G.J. Sullivan, J. Ohm, Woo-Jin Han; T. Wiegand, T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," Circuits and Systems for Video Technology, IEEE Transactions on , vol.22, no.12, pp.1649,1668, Dec. 2012.

[14] Yu-Wen Huang et al., "Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 796–799, May 2003.

[15] Swee Yeow Yap et al., "A VLSI Architecture for Variable Block Size Video Motion Estimation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 51, no. 7, pp. 384 – 389, July 2004.

[16] M. Haller et al., "Robust global motion estimation using motion vectors of variable size blocks and automatic motion model selection," in *IEEE International Conference on Image Processing (ICIP)*, pp. 737 –740, Sept. 2010.

[17] Chia-Chun Lin et al., "PMRME: A Parallel Multi-Resolution Motion Estimation Algorithm and Architecture for HDTV Sized H.264 Video Coding," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 385–388, April 2007.

[18] M. E. Sinangil, V. Sze, M. Zhou, A. P. Chandrakasan, "Memory Cost vs. Coding Efficiency Trade-Offs for HEVC Motion Estimation Engine," *IEEE International Conference on Image Processing (ICIP)*, pp. 1533-1536 , Sep. 2012.

[19] "Taiwan Semiconductor Manufacturing Company Limited." [Online]. Available: http://www.tsmc.com/english/dedicatedfoundry/technology/65nm.htm.

[20] M. E. Sinangil, V. Sze, M. Zhou, A. P. Chandrakasan, "Hardware-Aware Motion Estimation Search Algorithm Development for High-Efficiency Video Coding (HEVC) Standard," *IEEE International Conference on Image Processing (ICIP)*, pp. 1529-1532, Sep. 2012.

Mahmut E. Sinangil (S'06–M'12) received the B.Sc. degree in electrical and electronics engineering from Bogazici University, Istanbul, Turkey, in 2006, and the S.M. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2008 and 2012 respectively.

Since July 2012, he has been a Research Scientist in the Circuits Research Group at NVIDIA. His research interests include low-power and application specific on-chip memories targeted towards graphics applications.

Dr. Sinangil was the recipient of the Ernst A. Guillemin Thesis Award at MIT for his Master's thesis in 2008, co-recipient of 2008 A-SSCC Outstanding Design Award and recipient of the 2006 Bogazici University Faculty of Engineering Special Student Award.

Vivienne Sze (S'06–M'12) received the B.A.Sc. (Hons) degree in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 2004, and the S.M. and Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2006 and 2010 respectively. She received the Jin-Au Kong Outstanding Doctoral Thesis Prize, awarded for the best Ph.D. thesis in electrical engineering at MIT in 2011.

Since August 2013, she has been with MIT as an Assistant Professor in the Electrical Engineering and Computer Science Department. Her research interests include energy efficient algorithms and architectures for portable multimedia applications. From September 2010 to July 2013, she was a Member of Technical Staff in the Systems and Applications R&D Center at Texas Instruments (TI), Dallas, TX, where she designed low-power algorithms and architectures for video coding. She also represented TI at the international JCT-VC standardization body developing HEVC, the next generation video coding standard. Within the committee, she was the primary coordinator of the core experiment on coefficient scanning and coding.

Dr. Sze was a recipient of the 2007 DAC/ISSCC Student Design Contest Award and a co-recipient of the 2008 A-SSCC Outstanding Design Award. She received the Natural Sciences and Engineering Research Council of Canada (NSERC) Julie Payette fellowship in 2004, the NSERC Postgraduate Scholarships in 2005 and 2007, and the Texas Instruments Graduate Woman's Fellowship for Leadership in Microelectronics in 2008. In 2012, she was selected by IEEE-USA as one of the "New Faces of Engineering".

Minhua Zhou (S'06–M'12) received his B.E. degree in Electronic Engineering and M.E. degree in Communication & Electronic Systems from

Shanghai Jiao Tong University, Shanghai, P.R. China, in 1987 and 1990, respectively. He received his Ph.D. degree in Electronic Engineering from Technical University Braunschweig, Germany, in 1997. He received Rudolf-Urtel Prize 1997 from German Society for Film and Television Technologies in recognition of Ph.D. thesis work on "Optimization of MPEG-2 Video Encoding".

From 1993 to 1998, he was a Researcher at Heinrich-Hertz-Institute (HHI) Berlin, Germany. Since 1998, he is with Texas Instruments Inc, where he is currently a research manager of video coding technology. His research interests include video compression, video pre- and post-processing, end-to-end video quality, joint algorithm and architecture optimization, and 3D video.

Anantha P. Chandrakasan (M'95–SM'01–F'04) received the B.S, M.S. and Ph.D. degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1989, 1990, and 1994 respectively. Since September 1994, he has been with the Massachusetts Institute of Technology, Cambridge, where he is currently the Joseph F. and Nancy P. Keithley Professor of Electrical Engineering.

He was a co-recipient of several awards including the 1993 IEEE Communications Society's Best Tutorial Paper Award, the IEEE Electron Devices Society's 1997 Paul Rappaport Award for the Best Paper in an EDS publication during 1997, the 1999 DAC Design Contest Award, the 2004 DAC/ISSCC Student Design Contest Award, the 2007 ISSCC Beatrice Winner Award for Editorial Excellence and the ISSCC Jack Kilby Award for Outstanding Student Paper (2007, 2008, 2009). He received the 2009 Semiconductor Industry Association (SIA) University Researcher Award. He is the recipient of the 2013 IEEE Donald O. Pederson Award in Solid-State Circuits.

His research interests include micro-power digital and mixed-signal integrated circuit design, wireless microsensor system design, portable multimedia devices, energy efficient radios and emerging technologies. He is a co-author of Low Power Digital CMOS Design (Kluwer Academic Publishers, 1995), Digital Integrated Circuits (Pearson Prentice-Hall, 2003, 2nd edition), and Sub-threshold Design for Ultra-Low Power Systems (Springer 2006). He is also a co-editor of Low Power CMOS Design (IEEE Press, 1998), Design of High-Performance Microprocessor Circuits (IEEE Press, 2000), and Leakage in Nanometer CMOS Technologies (Springer, 2005).

He has served as a technical program co-chair for the 1997 International Symposium on Low Power Electronics and Design (ISLPED), VLSI Design '98, and the 1998 IEEE Workshop on Signal Processing Systems. He was the Signal Processing Sub-committee Chair for ISSCC 1999-2001, the Program Vice-Chair for ISSCC 2002, the Program Chair for ISSCC 2003, the Technology Directions Sub-committee Chair for ISSCC 2004-2009, and the Conference Chair for ISSCC 2010-2012. He is the Conference Chair for ISSCC 2013. He was an Associate Editor for the IEEE Journal of Solid-State Circuits from 1998 to 2001. He served on SSCS AdCom from 2000 to 2007 and he was the meetings committee chair from 2004 to 2007. He was the Director of the MIT Microsystems Technology Laboratories from 2006 to 2011. Since July 2011, he is the Head of the MIT EECS Department..