Approximate-Computing Architectures for Motion Estimation in HEVC

Approximate-Computing Architectures for Motion Estimation in HEVC

Alberto Paltrinieri*, Riccardo Peloso*, Guido Masera*, Muhammad Shafique† and Maurizio Martina*
*Department of Electronics and Telecommunications, Politecnico di Torino, Italy
†Institute of Computer Engineering, Vienna University of Technology (TU Wien), Austria
Email: alberto.paltrinieri@studenti.polito.it, riccardo.peloso@polito.it,
guido.masera@polito.it, maurizio.martina@polito.it, muhammad.shafique@tuwien.ac.at

***Abstract*—Video compression deeply relies on motion estimation but this crucial step requires power-hungry computational resources. Starting from an existing architecture capable of calculating a high number of Sum of Absolute Differences (SADs), three possible locations for approximate adders are chosen. At each location, different types of approximate adders are implemented and the results are analyzed in terms of error and power saving.**

I. Introduction

The High Efficiency Video Coding (HEVC, also called H.265) is the latest standard in video compression developed by the Moving Picture Experts Group (MPEG). This new compression scheme is able to double the compression efficiency with respect to its predecessor, the Advanced Video Coding (AVC or H.264). The HEVC heavily exploits the inter-frame (time-based) and intra-frame (spatial-based) redundancies to reduce the amount of data to be stored or transferred. The techniques used are the Motion Estimation (ME) and the Motion Compensation (MC), which try to estimate a suitable Motion Vector (MV) that minimizes the prediction error. This is done by computing the error, usually through the Sum of Absolute Differences (SAD) operation, between the current block to be encoded and the surrounding pixels in already encoded frames. If a match is found, only the MV and the corrections to perform the prediction are encoded. This process effectively reduces the total amount of compressed data but it requires a very high performance ME engine. The best match is defined as the one producing the minimum SAD value found. As a consequence, it is important to build an hardware accelerator of the SAD function, able to process several data concurrently. The ideal architecture should be able to sustain a high throughput while consuming a reduced amount of power: the ME is one of the most computation intensive tasks to be performed by the video encoder, indeed. In [1] an efficient VLSI architecture for the computation of the SAD in HEVC has been proposed. Such an architecture used the partial distortion elimination (PDE) technique to speed up the processing by avoiding useless computations. Stemming from [1], this current work aims to study the effect of approximate adders with the objective of saving power consumption while keeping the error as small as possible.

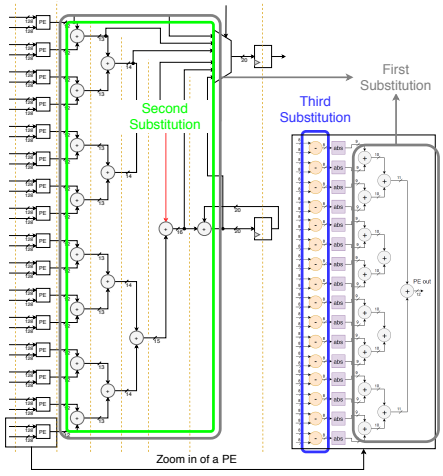

Fig. 1: SAD architecture where the three substitution regions are highlighted.

II. SAD architecture

This section aims to briefly review the basic SAD architecture in [1] in order to define where it is possible to employ approximate adders to save power. The SAD operation for an $N \times M$ block of pixels is defined as

$$SAD = \sum_{i=1}^{N} \sum_{j=1}^{M} |C(i,j) - R(i,j)| \tag{1}$$

where C is the current block and R is the reference block. In the worst case ($N = M = 64$), the SAD computation requires to read 4096 samples of the current Prediction Unit (PU) and 4096 samples of the reference PU. Then, according with (1), it carries out the difference between 4096 samples, 4096 absolute values and the 4095 additions. Such a structure can be strongly parallelized to speed up the execution. This is usually obtained by processing several pairs of samples concurrently and resorting to a tree-adder. One of the problems that arises from a parallel architecture is the memory bandwidth limit. The memory bandwidth is responsible for throughput reduction in motion estimation, in particular when dealing with high resolution/quality video sequences [2]. The starting architecture considered in this work is the hardware structure proposed in [1], which computes the SADs on all the input Prediction Blocks (PBs) for every input dimension

up to 64x64. This result is obtained by properly storing current PU and reference PU samples into a DRAM. The DRAM is clocked at eight times the main clock frequency to provide the required number of pixels per clock cycle. A control unit dynamically reorders the input blocks as a vector to maximize the throughput. Then, 16 Processing Elements (PEs) are exploited to compute up to 256 differences and absolute values concurrently. Finally, the tree-adder combines partial results (see Fig. 1). Depending on the PU size a variable number of clock cycles (ranging from 1 to 16) is required o complete the computation of one SAD value. To speed up the computation, the PDE algorithm is employed: during the best match computation, if an adder returns a result greater than the current best SAD value there is no reason to keep going in the tree-adder computing as the final result will surely be discarded. This scheme needs some additional comparators in order to work properly, as detailed in [1].

III. Approximate computing

Among the video compression algorithms, approximate computing has become an emerging paradigm to reduce the power/energy consumption of the system and to reduce the computational load of a compressing procedure [3]–[5]. It exploits the resilience of the applications to speed up the data processing of the system and trading rate-distortion performance for power consumption. The main task of approximate computing is to relax signal processing tasks at different levels. Indeed, exploiting the tolerance of computing imprecision of the human eye imperceptibility [6], highly power-efficient implementations can be realized [7]. If the amount of computing imprecision is limited, the result is a negligible amount of perceptible visual change of the output sequence. Of course, the error introduced by the approximate arithmetic must not exceed a certain threshold, or the reduction in resolution can ruin the visual experience. The ME shows a certain level of resilience or elasticity for small computational errors [8]. Even if the MV found by approximate computing is not the best one, the result could be still acceptable in terms of compression ratio. For this reason, this current work embraces the approximate computing paradigm by substituting correct adders with approximate ones inside the SAD architecture proposed in [1] and depicted in Fig. 1. To deeply study the accelerator behavior, a crossed study was carried out, analyzing both precise and approximate adders in order to verify not just the approximation level (with consequent improvements introduced by the approximate adders) but also to test the adaptability of any kind of adder to the design. Furthermore, the position where the adders were substituted can not be neglected: three substitutions were analyzed to find a good blend of correct and approximate adders in the SAD architecture.

A. Approximate adders

The choice of the approximate adders has been carried out through literature analysis by investigating most of the possibilities. In brief, approximate adders can be classified in different classes [9]:

- Speculative adders
- Segmented adders
- Carry-Select adders
- Approximate full adders

The speculative adders exploit the longest sequence of 1 inside the propagate signal p for the parallel computing of long binary sequences. The segmented adders divide the bit sequence in several slices, addition is performed for each slice independently and, finally, slice results are combined, giving precedence to the most significant bits (MSBs). The carry select adders consist of several sub-adders which execute the the addition with both carry-in equal to 0 and equal to 1 concurrently, then the correct result is selected. Finally, approximate full adders divide the n-bit inputs into 2 parts, the MSB computation is correct, whereas the least significant bits (LSBs) are approximated and generate no carry.

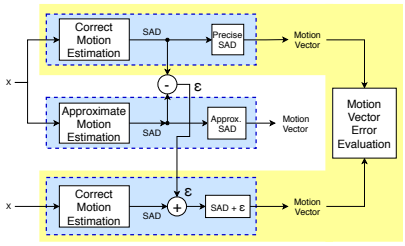

Fig. 2: Software model of the architecture, in blue the C++ part and in yellow the MATLAB part

IV. Approximate architecture

In this work the following correct and approximate adders have been considered and analyzed. For further details the reader can refer to [10] and [6].

- RCA: Ripple Carry Adder
- CLA: Carry Look-ahead Adder

The considered five approximate adders are:

- ACA: Almost Correct Adder (Speculative adders class)
- ACAA: Accuracy Configurable Approximate Adder (Segmented adders class)
- ETA-I (ETAI): Error-Tolerant Adder I (Approximate full adders class)
- LOA: Lower-OR Part Adder (Approximate full adders class)
- SCSA: Speculative Carry Select Adder (Carry-select adders class)

In order to analyze the effect of each type of adder in different positions inside the SAD architecture, the following three regions have been identified (as highlighted in Fig. 1):

1) Following the approach described in [4], all correct adders in the architecture, including the ones in the PEs and in the tree-adder have been replaced by the approximate ones. Only subtracters are kept correct. This configuration is highlighted by a gray box in Fig. 1. The

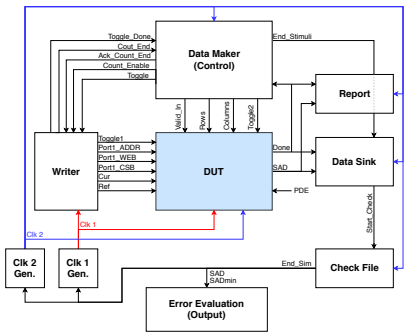

Fig. 3: Testbench used to verify the correct behavior of the implemented architecture

error introduced by this solution is rather high, being the highest among the tested substitutions, but it is the one providing the smallest delay.

2) Adders are substituted only outside the PEs, namely in the tree-adder. Indeed, experiments have shown that approximate adders inside PEs affect ME result more than approximate adders in the tree-adder. Simulations have shown that partial results along the tree-adder frequently fit in 12 bits, despite up to 20 bits are needed for correct computation. As a consequence, adders handling bits from 0 to 11 are kept correct, whereas the ones related to bits from 12 to 19 have been approximated. This configuration, shown as a green box in Fig. 1, reduces the error with some improvements in terms of delay.
3) Stemming from the idea presented in [11], correct subtracters inside each PE are replaced by approximate ones. This substitution generates an error that propagates through the the whole architecture. However, since there are up to 4096 differences, it produces a relevant speed-up. This substitution relies on low-error approximate adders to reduce error propagation and and it is shown as a blue box in Fig. 1.

V. Implementation

Then, the model has been embedded in a MATLAB environment as shown in Fig. 2. This strategy allows to extract the statistic of the error introduced by the approximate architecture (ϵ) and to analyze the effect on MVs for a large amount of data.

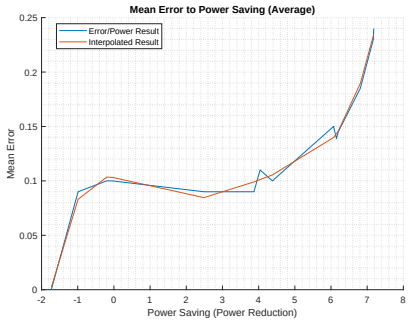

Fig. 4: MRED vs. power saving (mW)

	MRED	Energy = $\sum \vert \vert^2$
Correct SAD	0%	$1283544 \simeq 1.28 \cdot 10^6$
Approximate SAD	6.7%	$1282726 \simeq 1.28 \cdot 10^6$

TABLE I: Results of the motion vector error estimation with the substitutions applied

	RCA, CLA	LOA	ACA	ACAA	ETAI	SCSA
1^{st} sub.	0	0.265	0.198	0.203	0.226	0.181
2^{nd} sub.	0	0.164	0.114	0.108	0.145	0.097
3^{rd} sub.	0	0.139	0.087	0.084	0.137	0.050

TABLE II: MRED error for each substitution

A. Software implementation

Firstly, a C++ model has been developed, which describes the architecture of the SAD accelerator described in [1]. This model reproduces the bit-accurate behavior of the architecture. It reads an input video sequence in .yuv format and performs a full search of all the possible PUs in the preceding frame for all the frames of the video.

The error on MVs has been analyzed through the following metrics:

- **Error Distance**: $ED = |S' - S|$, where S' is the sum of the approximate adder and S is the sum of the accurate adder;
- **Mean Error Distance**: $MED = \frac{\sum_i ED_i}{N_{PU} \cdot N_{frames}}$, where N_{PU} is the number of PUs and N_{frames} is the total number of frames;
- **Relative Error Distance**: $RED = \frac{|S'-S|}{S}$;
- **Mean Relative Error Distance**: $MRED = \frac{\sum_i RED_i}{N_{PU} \cdot N_{frames}}$;
- **Percentage Error**: $E_{MRED}(\%) = MRED \cdot 100$
- **Total Percentage Error**: TPE(%) is the most common error metric: it represents the number of errors calculated for every SAD case divided by the total number of events, as a percentage. In other words, it highlights how many times the output SAD values differ in the case of correct or approximate adders;
- **Energy difference**: $E_n = \sum_{i=1}^{N} |PU_{current} - PU_{MCreference}|^2$, this is a different type of error metric because it works on the difference of energy between the reference PU and the motion compensated PU.

	1^{st} sub.	2^{nd} sub.	3^{rd} sub.
[1]	127.94 mW		
ACA	123.56 mW	128.14 mW	128.93 mW
ACAA	121.13 mW	123.90 mW	125.45 mW
CLA	129.67 mW	129.67 mW	128.18 mW
RCA	128.07 mW	128.02 mW	128.14 mW
ETAI	120.77 mW	121.87 mW	121.88 mW
LOA	120.76 mW	121.79 mW	121.74 mW
SCSA	124.07 mW	127.99 mW	128.78 mW

TABLE III: Extracted power consumption for all the possible implementations

As the $MRED$ and the E_n turned to be the most significant tools for measuring the error, the preliminary results obtained

	[12]	[13]	[14]	[2]	[8]	[1]	Proposed
Technology	65 nm	Virtex-5	Virtex-5	65 nm	45 nm	65 nm	65 nm
Max Block Size	64x64	64x64	64x64	32x32	64x64	64x64	64x64
Gate Count	434K	26.8K	63.2K	617K	30.9K	90K	$\sim$ 90K
Frequency [MHz]	720	190.785	348	350	497.66	160 *	160 *
Delay for each 64x64 Block	22.24 ns	167.73 ns	45.96 ns			156.25 ns	156.25 ns
Search Window	($\pm$27,$\pm$27)			($\pm$32,$\pm$32)		($\pm$64,$\pm$64)	($\pm$64,$\pm$64)

TABLE IV: Performance comparison of the proposed architecture with other works from the literature.

with the proposed environment are reported in Table I. Moreover, Fig. 4 shows MRED as a function of the achieved power saving. As it can be observed the first part of the graph is non-monotonic, showing that one can save 5 to 7 mW with a very small error.

B. Hardware implementation

After evaluating via software the various possibilities, the modified adders have been described in *VHDL*, placed in the SAD architecture and tested (see Fig. 3). The complete architecture has been synthesized on a 65 nm standard cell technology. Table II highlights the MRED values obtained by the various adders for the three substitutions. As it can be observed, LOA is always the solution leading to the highest MRED values, whereas SCSA features low MRED for all the considered cases.

VI. Results

After the logic synthesis the switching activity has been extracted for all the described implementations in order to measure power consumption. In terms of power saving, the ACAA, the ETAI and the LOA are the approximate adders that are most efficient. It is useful to see how, in average, the power relates to the approximation error, as in Figure 4. The curve is nonlinear and presents a plateau at mean error of about 0.1, which means that it is possible to mantain approximately constant this error while employing more aggressive approximate adders. Table III shows the power saving obtained with respect to the baseline structure. The approximate adders lead to up to about 7 mW of power saving. It is worth noting that the results differ from the ones in [1] as the approximate architecture is here dealing with a true video stream, i.e. handling a sequence of PU with different sizes, without forcing a specific block size.

Table IV shows a comparison with other implementations. The '*' states that the frequency of the two works are limited by the SRAM IP core speed. As it can be observed, the proposed architecture compares favorably with the other ones, featuring low complexity. Given that the 160 MHz clock frequency is a constraint of the architecture in [1], the proposed approximate architecture can be exploited to reduce the power consumption, as shown in Table III.

VII. Conclusion

This work showed that approximate adders reduce the overall power consumption of a SAD architecture for ME. In particular, choosing the proper blend of exact and approximated adders leads to a power efficient architecture, with limited quality loss.

References

[1] P. Selvo, M. Masera, R. Peloso, G. Masera, M. Shafique, and M. Martina, "An optimized partial-distortion-elimination based sum-of-absolute-differences architecture for high-efficiency-video-coding," in *APPLEPIES*, 2018.

[2] S. Park, B. G. Choi, I. G. Lim, H. il Park, and S. W. Kang, "An efficient motion estimation hardware architecture using modified reference data access(MRDAS) skip algorithm for high efficiency video coding(HEVC) encoder," in *2016 IEEE 6th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. IEEE, sep 2016.

[3] E. Nogues, D. Menard, and M. Pelcat, "Algorithmic-level approximate computing applied to energy efficient hevc decoding," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.

[4] W. El-Harouni, S. Rehman, B. S. Prabakaran, A. Kumar, R. Hafiz, and M. Shafique, "Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 1384–1389.

[5] M. Masera, M. Martina, and G. Masera, "Adaptive approximated dct architectures for hevc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 12, pp. 2714–2725, Dec 2017.

[6] Y. Jia, W. Lin, and A. A. Kassim, "Estimating just-noticeable distortion for video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 7, pp. 820–829, July 2006.

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

[8] R. Porto, L. Agostini, B. Zatt, M. Porto, N. Roma, and L. Sousa, "Energy-efficient motion estimation with approximate arithmetic," in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, oct 2017.

[9] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15. ACM, 2015.

[10] T. Land and M. D. Ercegovac, *Digital Arithmetic*. Elsevier, 2004. [Online]. Available: 10.1016/b978-1-55860-798-9.x5000-3

[11] R. Porto, L. Agostini, B. Zatt, M. Porto, N. Roma, and L. Sousa, "Energy-efficient motion estimation with approximate arithmetic," in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, Oct 2017, pp. 1–6.

[12] A. Medhat, A. Shalaby, and M. S. Sayed, "High-throughput hardware implementation for motion estimation in hevc encoder," in *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2015, pp. 1–4.

[13] V. N. Dinh, H. A. Phuong, D. V. Duc, P. T. K. Ha, P. V. Tien, and N. V. Thang, "High speed SAD architecture for variable block size motion estimation in HEVC encoder," in *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*. IEEE, jul 2016.

[14] A. Medhat, A. Shalaby, M. S. Sayed, M. Elsabrouty, and F. Mehdipour, "A highly parallel SAD architecture for motion estimation in HEVC encoder," in *2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, nov 2014.