Exploring Absolute Differences Arithmetic Operators for Power- and Area-Efficient SAD Hardware Architectures

Brunno A. Abreu¹, Mateus Grellert², Guilherme Paim³, Leandro M. G. Rocha⁴, Cláudio M. Diniz⁵, Eduardo A. C. da Costa⁶, Sergio Bampi⁷

¹,³,⁴,⁷ Graduate Program in Microelectronics (PGMicro) - Federal University of Rio Grande do Sul (UFRGS), Porto Alegre - Brazil
² Embedded Computing Lab - Federal University of Santa Catarina (UFSC), Florianópolis - Brazil
⁵,⁶ Graduate Program in Electronic Engineering and Computing - Catholic University of Pelotas (UCPel), Pelotas - Brazil
e-mail: {baabreu, gppaim, lmgrocha, bampi}@inf.ufrgs.br, mateus.grellert@ufsc.br, {claudio.diniz, eduardo.costa}@ucpel.edu.br

Abstract—This paper explores various structures of absolute operators and their employment in Sum of Absolute Differences (SAD) architectures to attain power-efficient SAD modules. Besides the analysis of several different models, we exploit the use of pipelining and the impact of varying block width to determine which versions scale better with the increase of input size. We have synthesized the architectures for ASIC CMOS technology using real input vectors obtained from a widely used video encoder application. The designs were compared with the default absolute operator (macrofunction) from the synthesis tool. The synthesis was performed using the ST 65nm standard cells library under two levels of optimization effort to verify how the circuit optimization heuristics implemented in the tool affect the results. By setting the effort to low, i.e., by minimizing the optimizations from the synthesis tool, we obtained power and area reductions of up to 17.04% and 25.61% respectively. By letting the tool optimize the architectures, we were able to reduce power and area in up to 6.66% and 16.06% respectively.

Index Terms—Video Coding, Hardware Architecture, Absolute, SAD.

I. INTRODUCTION

The latest advances achieved by the semiconductor industry in recent years have allowed multimedia applications to increase their quality considerably. Such improvements are essential when dealing with the increasing demands of these services. Digital videos are among the most critical types of applications, given the high amount of resources needed for managing them, especially when considering energy requirements for embedded systems to achieve real-time throughput.

The improvements achieved from the proposals implemented in latest video encoders led to an increase in the computational effort. For example, encoders compliant with the High Efficiency Video Coding (HEVC) standard are 1.2–3.2× more complex when compared to implementations of encoders compliant with its predecessor standard H.264/AVC [1]. Although such improvements handle compression and image quality demands, they also convey a critical issue related to the higher energy requirements to encode bitstreams compatible with this standard. A typical solution to deal with this issue is to develop dedicated hardware architectures for video coding stages, as they are more power-efficient than general-purpose processors, given that they are designed for specific applications.

Motion Estimation (ME) is a key component in video encoders, as it is responsible for exploiting temporal redundancies in video sequences. This stage is computed several times for each frame, making it one of the most costly steps in the encoding process. By using an HEVC encoder as a case study, it has been observed that the ME stage consumes more than 60% of the total encoding time [2]. A popular metric used in ME to calculate block similarity is the Sum of Absolute Differences (SAD). An analysis on how much SAD represents in the total encoding time is presented in Figure 1. This experiment was performed for different video resolutions and based on [3], showing that SAD takes a share of, on average, 22.4% of the total encoding time. Therefore, developing solutions that optimize SAD calculation in terms of power dissipation is an effective way to decrease the overall energy requirements of encoders.

SAD architectures are usually implemented with an adder tree, with its first level composed of subtractors and absolute operators. Related works on power-efficient solutions for SAD mostly focus on optimizing the adders used in the tree, disregarding the decisions on the subtractors and absolute operators. To the best of our knowledge, the literature lacks an exploration towards power-efficient absolute and subtractor operators in SAD context. This paper presents a design space exploration of the absolute and subtractor operators in SAD architectures. The contributions of this work are the following:

• Proposal of several architectures for performing the subtraction and absolute operations in SAD context, includ-

Fig. 1: Encoding time of several HEVC components for three video sequences [4].
ing a version that takes advantage of the carry-in inputs of subsequent adder tree units;

- Analysis of the scalability of each presented model by increasing the input width of the SAD architecture;

- Exploration of the best models with/without pipeline for varying input widths;

- Comparison of each proposal with the baseline operator from the synthesis tool under multiple levels of optimization effort.

Power dissipation and circuit area results were extracted, so that we could define the most suitable implementation for each input width size, and to find out whether the more efficient versions – in any of the two axes – vary with/without a pipeline. This work is an extended version of [5], which only analyzed architectures computing 8 input samples in parallel, whereas this paper analyzes architectures with 8, 16, 32 and 64 samples of parallel input processing. The architectures were synthesized for ASIC, using a low-power 65 nm commercial standard-cell library from ST, using real input values from the HEVC Test Model (HM) reference software [6], taking the gate and interconnection delays into account in the simulation. The results are compared with a model using the subtraction and absolute operator from the synthesis tool, for each respective combination of input width and usage of a pipeline.

This paper is organized as follows. Section II gives a background on the main concepts related to Integer Motion Estimation, including metrics and algorithms employed in this process. Section III presents the designed architectures, with varying input widths and pipeline stages. In section IV, the results are presented compared to the baseline subtractor and absolute operators from the synthesis tool. Section V concludes the paper, highlighting its main contributions.

II. BACKGROUND

A. Motion Estimation

ME is the stage of the video encoding process responsible for exploring the inherent temporal redundancies in video sequences in order to decrease the final size of the encoder bitstream – while attempting to maintain visual quality –, increasing the throughput of the encoding process. This is achieved by finding a block, in a previously encoded frame (reference frame), similar to the colocalized vector – pointing to the upper-left corner of the best candidate found in the IME. The most naïve BMA implementation is the one that searches for every block possibility inside the given search window, displacing one sample at a time. This is called Full Search (FS) and, even though this BMA finds the best optimal block inside the search window, this algorithm is too costly in terms of encoding time. Sub-optimal BMAs usually choose their search start inside the search window based on a Motion Vector Prediction (MVPred) algorithm. These algorithms search through a set of vectors (which point to candidate blocks), both temporally and spatially related to the block being encoded, and chooses the one with the highest similarity to begin the search.

B. Test Zone Search

The HM reference software implements a popular BMA, called Test Zone Search (TZS). This BMA represents a good trade-off between time and quality, i.e., even though it does not always find the optimal block inside the search window, it has a much smaller amount of block comparisons when compared to FS, making it less time-consuming and more energy efficient.

The default configuration of TZS is based on a diamond-shaped search, and is split into four subsequent stages:

- Search Vector Initialization: this is a stage in the beginning of the TZS implementation in HM, which checks whether the colocalized vector – pointing to the colocalized candidate block from the reference frame – or the vector resulting from the IME iteration of the $2N \times 2N$ partition are better than the current start vector, which was chosen by the MVPred stage. This defines a motion vector to be used as a starting point for the next stages;

- First Search (FiS): in this stage, the main diamond-shaped search is performed, starting from the position defined from the first step. The diamond pattern checks four, eight or sixteen candidates, depending on the number of iterations it demands, which is incremented after each diamond step. FiS ends when the algorithm does not find any better candidates for three iterations or when it reaches the borders of the search window;
• Raster Search: this step searches through the entire search area with a step of 5 samples between each candidate. This stage only executes if the vector generated so far by the algorithm is larger than a specific threshold, defined by the encoder;

• Refinement Search: the last stage also performs a diamond-shaped search. This stage consists of several iterations of FiS. This step, as well as the whole IME stage, stops when no better candidates have been found in an entire iteration. The best motion vector from this stage is chosen as the global motion vector of the IME execution, and it is ready for the FME to be applied.

These stages are briefly illustrated in Figure 2, which highlights the possible diamond and raster patterns, with a vector starting from the decision in the search vector initialization step. TZS implementation in HM has more stages than the ones described here; however, those are not executed in the default preset of the encoder.

C. Sum of Absolute Differences

Finding the similarity among pixel blocks requires a meaningful yet not overly complex metric. Among the existent metrics for that matter, the Sum of Absolute Differences (SAD) is the most popular one, when considering the IME stage. Its popularity is mainly because its calculation is simple (as shown in Equation 1): SAD adds the sum of the absolute values of the co-localized samples of the two blocks being compared.

\[
SAD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |O_{i,j} - R_{i,j}|
\]

(1)

Its simplicity is also reflected in hardware architectures, given that SAD can be implemented with a tree of adders, with its first level being composed of subtractors and absolute operators. The schematics of a SAD architecture is shown in Figure 3. The architecture calculates SAD for 8 samples and is composed by 8 subtractors and absolute operators, to obtain the absolute differences between the samples from original and reference blocks, and 7 adders that form the adder tree. An additional adder is placed at the output so that SAD for larger blocks may be accumulated, totaling 8 adders. The number of accumulations required to calculate SAD for an \(X \times Y\) block considering an architecture with an input width of \(N\) is given by Equation 2.

\[
\text{#Accumulations} = \frac{X \times Y}{N}
\]

(2)

As it can be seen from Figure 3 and from the previous description, parallelism is easily exploited in SAD architectures, given that most blocks can operate independently. As a result, SAD allows for several different configurations of input width and number of pipeline levels. In the literature, a wide range of different adder tree schemes and proposals can be found. This work, however, focuses on the design of power-efficient modules for the subtractors and absolute operators, which compose the first and second stages from Figure 3. To the best of our knowledge, very few works have made that kind of exploration on subtraction and absolute operators in the context of SAD architectures.

![Fig. 3: SAD Architecture; (a) Absolute Difference and (b) Adder Tree with accumulation; and the possible pipeline levels in red [5].](image)

Typical operators to subtract and calculate the absolute compute the 2’s complement of the input value and use its Most Significant Bit (MSB) to select the correct output, using it as the selector to a multiplexer. In this work, we analyze several well-known implementations along with our proposed architectures. Alternatively, many of the works regarding SAD architectures use the subtraction and absolute macrofunctions, and leave the implementation of this first level to the synthesis tool. Even though the tool usually performs well for general cases, some modifications may be done, some of which are proposed in this work, to improve speed or power of subtractors along with the absolute operators in SAD. One of our proposals is an architecture that takes advantage of the carry-in input of full adders to perform the ‘+1’ sum required to invert the value in the subtraction operation.

D. Related Work

Several works have been found in the literature focusing on the optimization of the SAD unit, which is also the main basis of our proposal. However, in most of the works, there is a shortage of details on absolute operators, as most of them target the adder tree.
Jehng et al. [7] propose an architecture for the FS algorithm. One of the processing elements is the absolute differences module, and it is implemented by inverting one of the inputs, adding it to the other one, and then the generated MSB is accumulated to this sum and determines whether the final sum is inverted. This model is also implemented in one of the architectures for this work.

Vassiliadis et al. [8] propose a SAD architecture, and it presents detailed information about absolute differences operator implementation. The proposal is based on determining the smallest value out of the two operators, inverting it, and adding a correction term to it. However, the paper does not explicitly present where the correction term is added. Therefore, even though the architecture could not be entirely replicated, a similar strategy has been proposed in our work, in which we add correction terms along with the adder tree, which will be presented in the next section.

Yap et al. [9] implement a ME architecture for the FS BMA. SAD architecture is included within the proposal, and the paper briefly specifies how its subtractor and absolute operators were implemented. However, no analysis seems to have been performed on why this model was employed. Moreover, the analysis considers previous and outdated video coding standards. Furthermore, the analysis considers an unfeasible BMA for real-time encoding of high-resolution videos.

Vanne et al. [10] propose a high-performance SAD architecture for ME. Even though the work is focused on previous standards as well, a more detailed analysis on the absolute difference operator is presented. Three absolute difference operators are considered in this paper [7, 8] in order to present an improved absolute difference module. The absolute differences is based on inverting one of the inputs, adding the values and maintaining or inverting the output based on the carry-out bit of the addition. Moreover, correction bits are required to be added in the compression array unit, to make up for the incorrect result of the current absolute operation. However, the work does not present any power dissipation results for the presented modules. We implemented similar approaches that use correction bits in the adder tree, which are presented in the next section.

Liu et al. [11] propose an optimization of SAD for H.264/AVC encoder. Its architecture implements the absolute differences module in a similar way to the one applied in [10]. Given that this version also does not present power dissipation results, we have implemented this model as one of our comparisons as well.

Dinh et al. [12] implement an architecture for the ME, considering the HEVC encoder. The architecture has an input width of 16 samples. The absolute differences operator employed in this work implements the two possible subtractions in parallel, and the MSB selects the correct one. However, power dissipation results are not presented. For this reason, this model is also implemented in our work for comparison purposes.

Our previous work [5] is one of the few works that perform a fairly detailed analysis on absolute operators. However, the paper only considers one video for its real input vectors, which may not accurately represent the general case for every benchmark. Furthermore, the analysis only considers one possible input width. This paper further extends this analysis by considering four different input width sizes, to verify whether the results for the best cases are maintained. Moreover, it is important to notice that the results from our previous work in [5] were obtained by performing the synthesis using Cadence Encounter RTL Compiler [13]. This work, despite including analyses for more sizes of input widths, uses Cadence Genus Synthesis Solution [14] for obtaining power and area results. Therefore, it is expected that the baseline models from the synthesis tool may achieve better results than the previous ones from Encounter. Additionally, given that some architectures presented the same power and area results, they were ignored in this work. Also, this work includes the results with low and high optimization efforts from the synthesis tool, to compare the generated architectures when the tools have lower or higher influence in the generation of the netlists.

Despite its popularity, the SAD is not an unanimity as a block comparison metric on video coding applications. Sum of Absolute Transformed Differences (SATD) is another used metric, specially in FME blocks. The SATD implementation is similar to the SAD. Given that both use absolute operators, some optimizations adopted on the latter can also be applied to former.

Soares et al. [15] and Cancellier et al. [16] both implement SATD architectures. However, even though the architectures implement absolute operators, no details are given on how they are implemented.

Several other works that implement SAD, SATD or ME architectures have been found. However, these works either do not specify how the absolute differences operator is implemented, or do not present power dissipation results. Moreover, some of the works that present detailed information on the absolute differences operator usually employ the same models between each other, seemingly without a methodological basis. This paper is mainly focused on providing a detailed analysis on the possible choices of absolute operators, for conventional SAD adder trees, for different input widths and varied pipeline usage so that future SAD works can clearly choose a specific model. A summarized analysis on SAD related works is presented in Table I.

III. DESIGNED ARCHITECTURES

This work considers four different schemes to calculate the absolute differences operation, three of which are based on architectures found in the literature or new proposals, which are depicted in Figure 4. The remaining operator consists in the use of the abs macrofunction from the synthesis tool, whose implementation is not known beforehand. Furthermore, four different input widths are considered: 8, 16, 32 and 64 samples per block (original and candidate), which equals 128, 512, 2048 and 8192 bits. We also consider the architectures without and with full pipeline. Lastly, two optimization effort levels (low and high) are also considered, which determine the degree of influence that the synthesis tool will have on the RTL descriptions. These variations result in a total of 64 different architectures being evaluated in this work.
In Figure 4, two architectures, (a) and (b), work independently from the adder tree with which they are attached, whereas architecture (c) has an interdependency with the adder tree. Hence, versions (a) and (b) can be used in a wider variety of applications, e.g., operations that do not add the partial absolute values.

The dependency of architecture (c) occurs due to the fact that it propagates the most significant bit of the subtractor to the adder tree, connecting each one of them to each carry-in input of the adders. In addition to that, it is noticeable that the (c) version requires that the adders in the tree have enough carry-in inputs for the operation to work in an exact manner. Put differently, considering each adder consisting of one carry-in – such as a default implementation from the synthesis tool, which is employed here –, the number of adders after the subtractions must be equal to the number of subtractors.

Architecture (a) performs two parallel subtractions, calculating $O_{i,j} - R_{i,j}$ and $R_{i,j} - O_{i,j}$, and defining the correct operation by using the MSB of one of the subtractions, with a multiplexer. Architecture (b) implements a XOR gate along with an additional adder to perform the absolute operation. Architecture (c) works similarly to (b) but does not require the additional adder, as it uses the carry-in inputs of the remaining adders in the adder tree to make up for the sum that it would need to perform the input inversion.

### IV. Synthesis Results and Discussions

All the SAD architectures, with varying input bit-width and pipeline usage, were described in VHDL, being implemented according to each absolute operator presented. The frequency chosen was based on the input width of 8 samples, from a throughput analysis previously performed in [3] to determine a target frequency for a real-time scenario while minimizing the quality degradation. This analysis was based on the x265 software [30], which is an optimized HEVC implementation focused on a faster execution, and concluded that, for an $8 \times 1$ SAD architecture encoding Full HD (FHD) video sequences at 30 fps, the target frequency should be around 133 MHz, with a negligible quality degradation when compared to the default preset. This quality loss, however, is not a consequence of this work; this frequency was only chosen to illustrate a real-time and high-throughput case in which the absolute architecture could work. With that in mind, when considering the SAD architectures with 16, 32 and 64 input samples, the architecture has to run at 66.6 MHz, 33.3 MHz and 16.6 MHz, respectively.

The architectures were synthesized using Cadence Genus Synthesis Solution [14], considering both low and high optimization efforts. To obtain more precise switching activities for the application we are targeting, we extracted the inputs of SAD executions by running the software encoder using benchmark video sequences. This was employed because the default estimation of the switching activity does not lead to accurate results. Some of the employed sequences are based on the recommendations from the Common Test Conditions (CTC) [31]. The 4K video sequences were obtained...
from the Ultra Video Group [32]. The sequences used were BQTerrace, Cactus, Kimono, ParkScene, BasketballDrive, HoneyBee, Jockey, ReadySteadyGo, and YachtRide. As the primary usage of SAD is for video applications, using input vectors from benchmark video sequences is the best approach to obtaining precise power dissipation values. Aiming for unbiased results, we started the extraction of the inputs in a random frame of each video sequence.

First, we evaluated the differences in the dynamic power dissipation considering the different video sequences and for two different Quantization Parameters (QP), for each video sequence. This analysis was performed for the aforementioned video sequences and for QPs of 22 and 32, and are presented in Figure 5.

![Fig. 5: Effects of varying video sequences and QPs in the dynamic power dissipation.](image)

As it can be seen from Figure 5, no significant differences have been observed from the varying QPs, as some of the video sequences present higher power dissipation results for QPs 22 and 32. The analysis shows that the only significant differences lie on the textures of the videos. Therefore, for the remaining analysis, we only present the average results using the QP of 32, as the characteristics would be fairly similar with any of the QPs tested.

The power dissipation results for the 8-, 16-, 32- and 64-input SAD architectures, for low and high optimization efforts, considering full pipeline and no pipeline for each of them, are presented in Tables II and III. The results are an average of the video sequences previously mentioned. All the results are compared w.r.t. to the baseline implementation (“abs” macrofunction from the synthesis tool). Additionally, Table IV presents the results in terms of absolute circuit area and relative gate count, based on 2-input NANDX1 gates. We also show the total area savings w.r.t. to the same baseline (tool) operator.

As it can be seen in Tables II, III and IV, the optimizations performed by the synthesis tool when changing the optimization effort from low to high was significant. Apart from a few exceptions, the total power results presented with low effort were higher than in the high effort scenario. Likewise, in terms of area, every result under high effort conditions presented smaller gate count values when compared to their counterparts in the low effort scenario. Also, these values differ from the ones obtained in our previous work [5]. Since the synthesis tool employed in that work is a predecessor of Cadence Genus Synthesis Solution, the tool performed different optimizations and designed the netlist in a different manner.

According to the results in Tables II, III, IV, we obtained better results in several situations, especially when considering area savings, when compared to the baseline model from the synthesis tool. Reducing area is important to reduce the relative cost of a chip die. On consumer electronic circuits which are heavily based on System-on-Chips (SoC), saving area results in more functions being integrated into the same die. Further, smaller chips helps reducing the overall production costs per unit, which has a considerable impact on the industry-level. Also, reduced circuit area increases the number of dies per wafer, thus increasing yield. The area savings are higher for the (c) version in every scenario. Less savings are obtained when pipeline is used, probably because registers reduce the relative area spent on combinational logic. Also, in the versions with full pipeline, by increasing the input width, we have observed better area reductions, especially in model (c). When considering the low effort scenario, we obtained reductions of up to 25.61% without any pipeline and up to 21.31% with pipeline, both in the units with 64 samples. If we set the tool optimizations to a high effort, we obtain less significant reductions: up to 16.06 without pipeline (8 samples) and up to 12.47% with pipeline (64 samples).

In terms of power for the low effort scenario, we achieved power reductions of up to 17.04% when compared to the baseline version from the synthesis tool, in the case of a SAD architecture of 8 samples of input bit-width, without any pipeline. Even though the savings are not as significant in the pipeline version – mainly due to the aforementioned reason that the relative area of combinational logic is smaller when increasing the number of registers –, reductions of up to 5.19% were obtained in the version with full pipeline and 16 samples. However, the obtained power reductions are still significant, given that SAD is one of the main modules employed in the whole encoding process, and it is required several times for each PU being encoded in a single frame. Therefore, even small savings in power dissipation are significant for improving the overall power savings of an encoder, when considering SAD. The only cases in low effort optimization that have presented worse results than the baseline are with the (a) version.

As for the high effort cases, the results were modest, with reductions of up to 6.66% and 0.61% without and with pipeline, respectively, in version (c) with 8 samples. In this scenario, version (b) only reduced power in two cases (both without pipeline) and version (a) did not present any reductions w.r.t. the baseline model.

Although the proposals for full pipeline implementations provided less significant reductions, gains were still obtained in almost every scenario when employing version (c), even in applying high effort optimization. This is an important finding, and it shows that related works that are focusing on architectures using SAD should implement the dedicated version whose subtractor and absolute operators depend on the rest of the adder tree through the carry-in inputs. In only one single scenario, which is the full pipeline version in the architecture with 64 samples in the high effort optimization, the “abs” macrofunction from the synthesis tool has presented better power results than proposal (c).
Table II. Power Dissipation Synthesis Results (µW) of the Different SAD Architectures with a low synthesis optimization effort.

| Width (samples) | Version | Leakage | Dynamic | Total | Reduction | Leakage | Dynamic | Total | Reduction |
|-----------------|---------|---------|---------|-------|-----------|---------|---------|-------|-----------|
| 8 samples       | Tool    | 3.93    | 589.71  | 593.63| -         | 3.05    | 709.69  | 712.73| -         |
|                 | a       | 2.86    | 525.29  | 528.15| 11.03%    | 3.00    | 720.53  | 723.53| -1.51%    |
|                 | b       | 3.71    | 550.47  | 554.18| 6.65%     | 2.94    | 704.61  | 707.55| 0.73%     |
|                 | c       | 2.83    | 489.62  | 492.45| 17.04%    | 2.83    | 687.88  | 690.72| 3.09%     |
| 16 samples      | Tool    | 3.97    | 508.45  | 512.43| -         | 5.73    | 726.36  | 732.09| -         |
|                 | a       | 3.99    | 534.32  | 538.31| -5.05%    | 5.65    | 742.68  | 748.33| -2.22%    |
|                 | b       | 3.71    | 550.47  | 554.18| 6.65%     | 2.94    | 704.61  | 707.55| 0.73%     |
|                 | c       | 3.56    | 471.76  | 475.32| 7.24%     | 5.19    | 688.90  | 694.10| 5.19%     |
| 32 samples      | Tool    | 7.70    | 506.45  | 514.15| -         | 11.28   | 716.88  | 728.16| -         |
|                 | a       | 7.61    | 534.29  | 541.90| -5.40%    | 11.13   | 739.33  | 750.47| -3.06%    |
|                 | b       | 6.94    | 504.95  | 511.89| 0.44%     | 10.55   | 692.67  | 703.22| 3.43%     |
|                 | c       | 6.30    | 476.46  | 482.77| 6.10%     | 10.21   | 680.81  | 691.02| 5.10%     |
| 64 samples      | Tool    | 15.17   | 393.92  | 409.09| -         | 11.28   | 716.88  | 728.16| -         |
|                 | a       | 14.98   | 388.83  | 403.81| 1.29%     | 22.09   | 703.01  | 704.09| 0.80%     |
|                 | b       | 13.64   | 388.88  | 402.52| 1.61%     | 20.92   | 563.51  | 584.43| 4.03%     |
|                 | c       | 12.37   | 370.96  | 383.34| 6.30%     | 20.24   | 558.10  | 578.34| 5.03%     |

Table III. Power Dissipation Synthesis Results (µW) of the Different SAD Architectures with a high synthesis optimization effort.

| Width (samples) | Version | Leakage | Dynamic | Total | Reduction | Leakage | Dynamic | Total | Reduction |
|-----------------|---------|---------|---------|-------|-----------|---------|---------|-------|-----------|
| 8 samples       | Tool    | 2.19    | 475.83  | 478.02| -         | 2.68    | 656.42  | 659.10| -         |
|                 | a       | 2.32    | 488.24  | 490.56| -2.63%    | 2.87    | 701.07  | 703.94| -6.80%    |
|                 | b       | 2.54    | 474.43  | 476.98| 0.22%     | 2.74    | 662.88  | 665.62| -0.99%    |
|                 | c       | 2.16    | 444.02  | 446.17| 6.66%     | 2.70    | 652.35  | 655.05| 0.61%     |
| 16 samples      | Tool    | 3.26    | 478.37  | 481.64| -         | 5.04    | 677.63  | 682.67| -         |
|                 | a       | 3.61    | 530.32  | 533.93| -10.86%   | 5.39    | 720.97  | 726.36| -6.40%    |
|                 | b       | 3.40    | 485.50  | 488.89| -1.51%    | 5.19    | 686.03  | 691.23| -1.25%    |
|                 | c       | 3.18    | 452.66  | 455.84| 5.36%     | 5.10    | 674.12  | 679.22| 0.51%     |
| 32 samples      | Tool    | 6.22    | 477.69  | 483.91| -         | 9.88    | 668.61  | 678.49| -         |
|                 | a       | 6.99    | 541.36  | 548.36| -13.32%   | 10.53   | 713.25  | 723.78| -6.68%    |
|                 | b       | 6.53    | 484.73  | 491.25| -1.52%    | 10.18   | 676.28  | 686.45| -1.17%    |
|                 | c       | 6.09    | 459.21  | 465.31| 3.84%     | 10.00   | 667.21  | 677.21| 0.19%     |
| 64 samples      | Tool    | 12.26   | 371.22  | 383.48| -         | 19.55   | 545.41  | 564.96| -         |
|                 | a       | 13.62   | 398.04  | 411.65| -7.35%    | 20.68   | 568.44  | 589.12| -4.28%    |
|                 | b       | 12.80   | 366.22  | 379.02| 1.16%     | 20.15   | 548.36  | 568.51| -0.63%    |
|                 | c       | 11.93   | 352.76  | 364.69| 4.90%     | 19.80   | 547.44  | 567.23| -0.40%    |
Table IV. Circuit area, Gate Count Results for both low and high optimization efforts.

| W | Ver. | Low Effort | High Effort |
|---|---|---|---|
| | Tool | (µm²) | kGates | Red. | (µm²) | kGates | Red. |
| | Tool | 16 | 8 samples | 25.61% | 33956.76 | 6174.42 | 4.10 |
| | Tool | 16 | 22 samples | 21.79% | 5068.03 | 6357.23 | 0.69 |
| | Tool | 16 | 64 samples | 21.86% | 11876.63 | 7123.12 | 0.92 |
| | Tool | 64 | 8 samples | 12.37% | 31601.12 | 4.51 |
| | Tool | 64 | 32 samples | 12.19% | 49436.52 | 3.44 -32.37% 20053.40 | 45368.98 | 6953.79 |
| | Tool | 64 | 64 samples | 12.47% | 38634.11 | 3.48% | 42132.68 |
| | Tool | 64 | 256 samples | 11.65% | 9039.42 | 9.21% |
| | Tool | 64 | 1024 samples | 11.36% | 6547.79 | 9.62% |
| | Tool | 64 | 4096 samples | 11.24% | 6375.52 | 9.74% |
| | Tool | 64 | 16384 samples | 11.02% | 5598.94 | 21.79% |
| | Tool | 64 | 65536 samples | 10.93% | 10039.1 | 1.36 |
| | Tool | 64 | 262144 samples | 10.92% | 11760.73 | 1.35 |
| | Tool | 64 | 1048576 samples | 10.92% | 9881.14 | 1.29 |
| | Tool | 64 | 4194304 samples | 10.91% | 9039.42 | 1.14 |

1 Input Width

V. CONCLUSIONS

This work presented a detailed design space exploration of absolute operators in SAD architectures, given that there was a lack of a thorough analysis of such operators. We implemented several different versions of absolute architectures, by varying four different absolute operator versions – based on works found in the literature –, four possible input width sizes (8, 16, 32 and 64 samples), the use or not of a full pipeline, and varying the optimization effort of the synthesis tool from low to high. The architectures were synthesized in a 65 nm standard cell technology using a precise power extraction methodology using real-input vectors from random frames of benchmark video sequences, at different operating frequencies for each input width, halving its value when the width was doubled. The models were compared to the baseline absolute operator from the synthesis tool. The results have shown that the absolute operator is relevant for the total power dissipation of the SAD module, given the significant variations obtained. For the low effort results, we obtained power reductions of up to 17.04% (8 samples version) and 5.19% (version with 16 samples) for no pipeline and full pipeline, respectively. In terms of area, results of up to 25.61% were obtained without pipeline for this same optimization effort, and up to 21.31% with pipeline, both for versions with 64 samples. When applying a high optimization effort in the tool, the results were more modest, and the best reductions were for the units with 8 samples: up to 6.66% and 0.61% without and with pipeline, respectively. The area reductions were also less significant, with 16.06% as the best scenario without pipeline, for the unit with 8 samples, and 12.47% with pipeline in the version with 64 samples. Even though the reductions in the versions with pipeline were less significant than in the same models without pipeline, the designed architectures (b) and (c) still managed to reduce area in every scenario w.r.t. the baseline version from the tool.

VI. ACKNOWLEDGEMENTS

The authors would like to thank CNPq, IFRS and Capes Brazilian agencies for financial support to our research.

REFERENCES

[1] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard,” IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1649–1668, Dec 2012.
[2] M. Grellert, S. Bampi, and B. Zatt, “Complexity-scalable HEVC encoding,” in 2016 Picture Coding Symposium (PCS), Dec 2016, pp. 1–5.
[3] B. Silveira, G. Paim, B. Abreu, M. Grellert, C. M. Diniz, E. A. C. da Costa, and S. Bampi, “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, vol. 64, no. 12, pp. 3126–3137, Dec 2017.
[4] B. Silveira, B. Abreu, G. Paim, M. Grellert, R. Ferreira, C. Diniz, E. Costa, and S. Bampi, “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), Dec 2017, pp. 499–493.
[5] B. Abreu, G. Paim, M. Grellert, B. Silveira, C. Diniz, E. Costa, and S. Bampi, “Exploiting absolute arithmetic for power-efficient sum of absolute differences,” in 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Dec 2017, pp. 522–525.
[6] “HEVC Test Model (HM) v. 16.7,” http://hevc.hhi.fraunhofer.de.
[7] Y. Jehng, L. Chen, and T. Chiueh, “An efficient and simple VLSI tree architecture for motion estimation algorithms,” IEEE Transactions on Signal Processing, vol. 41, no. 2, pp. 889–900, Feb 1993.
[8] S. Vassiliadis, E. A. Hakkenes, J. S. S. M. Wong, and G. G. Pechanek, “The sum-absolute-difference motion estimation accelerator,” in Proceedings. 24th EUROMICRO Conference (Cat. No.98EX204), vol. 2, Aug 1998, pp. 559–566 vol.2.
[9] S. Y. Yap and J. V. McCanny, “A VLSI architecture for advanced video coding motion estimation,” Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors, vol. 2003-Janua, pp. 293–301, 2003.
[10] J. Vanne, E. Abo, T. D. Hamalainen, and K. Kussinlina, “A High-Performance Sum of Absolute Difference Implementation for Motion Estimation,” IEEE Transactions on Circuits Systems for Video Technology, vol. 16, no. 7, pp. 876–883, July 2006.
[11] Z. Liu, S. Goto, and T. Ikenaga, “Optimization of Propagate Partial SAD and SAD tree motion estimation hardwired engine for H.264,” in 2008 IEEE International Conference on Computer Design, Oct 2008, pp. 328–333.

[12] 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), July 2016, pp. 195–198.

[13] C. D. Systems, “ Encounter RTL Compiler Synthesis Flows,” http://www.cadence.com.

[14] ———, “Cadence EDA tools,” http://www.cadence.com.

[15] L. B. Soares, C. M. Diniz, E. A. C. da Costa, and S. Bampi, “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), Aug 2016, pp. 1–6.

[16] L. H. Cancellier, A. B. Brascher, I. Seidel, and J. L. Guntzel, “Energy-efficient Hadamard-based SATD architectures,” in 2014 27th Symposium on Integrated Circuits and Systems Design (SBCCI), Sep. 2014, pp. 1–6.

[17] M. Porto, A. Silva, S. Almeida, E. Costa, and S. Bampi, “Motion Estimation Architecture Using Efficient Adder-Compressors for HDTV Video Coding.” Journal of Integrated Circuits and Systems, vol. 5, no. 1, pp. 78–88, Mar 2010.

[18] F. L. Walter, C. M. Diniz, and S. Bampi, “Synthesis and comparison of low-power high-throughput architectures for SAD calculation,” in 2011 IEEE Second Latin American Symposium on Circuits and Systems (LASCAS), Feb 2011, pp. 1–4.

[19] P. Nalluri, L. N. Alves, and A. Navarro, “A novel SAD architecture for variable block size motion estimation in HEVC video coding,” in International Symposium on System on Chip (SoC), 2013.

[20] T. L. D. Dang, I. J. Chang, and J. Kim, “a-SAD: Power Efficient SAD Calculator for Real time H.264 Video Encoder Using MSB-Approximation Technique,” Proceedings of the 2014 international symposium on Low power electronics and design - ISLPED ’14, vol. 1, no. c, pp. 259–262, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2627369.2627650

[21] L. Yufei, F. Xiubo, and W. Qiu, “A High-Performance Low Cost SAD Architecture for Video Coding,” IEEE Transactions on Consumer Electronics, vol. 53, no. 2, pp. 535–541, 2007.

[22] C. Diniz, J. Altermann, E. Costa, and S. Bampi, “Performance Enhancement of H.264/AVC Intra Frame Prediction Hardware Using Efficient 4-2 and 5-2 Adder-compressors,” in SBCCI, 2010, pp. 157–162.

[23] L. Seidel, B. G. de Moraes, A. B. Brascher, and J. L. Guntzel, “On the impacts of pel decimation and High-Vt/Low-Vdd on SAD calculation,” in SBCCI, Sept 2013, pp. 1–6.

[24] L. Seidel, A. B. Brascher, and J. L. Guntzel, “Combining Pel Decimation with Partial Distortion Elimination to increase SAD energy efficiency,” in PATMOS, Sept 2015, pp. 177–184.

[25] P. Nalluri, L. N. Alves, and A. Navarro, “High speed SAD architectures for variable block size motion estimation in HEVC video coding,” in ICIP, Oct 2014, pp. 1233–1237.

[26] N. N. Shah, K. R. Agarwal, and H. M. Singapuri, “Implementation of sum of absolute difference using optimized partial summation term reduction,” in 2013 International Conference on Advanced Electronic Systems (ICAES), Sep. 2013, pp. 192–196.