Exploring Optimized Hadamard Methods to Design Energy-Efficient SATD Architectures

Luiz Henrique Cancellier[1], Ismael Seidel[2], André Beims Brascher[3], Jose Luis Guntzel[4], Luciano Agostini[5]

[1, 2, 3, 4] ECL – Embedded Computing Lab., Department of Informatics and Satistics, UFSC – Federal University of Santa Catarina, Florianópolis, Brazil
[1, 5]GACI – Group of Architectures and Integrated Circuits, UFPeL – Federal University of Pelotas, Brazil
e-mail: {l.h.cancellier, andre.brascher}@grad.ufsc.br, ismaelseidel@inf.ufsc.br, j.guntzel@ufsc.br, agostini@inf.ufpel.edu.br

ABSTRACT

State-of-the-art video coding tools are submitted to severe performance and energy consumption requirements resulting from high complexity of video standards and from limited energy budgets of portable mobile devices. While providing most of the compression gains, inter frame and intra frame prediction techniques are the most demanding steps, since they compare a huge number of blocks. In such a process, the similarity metric employed affects both the quality of compression and the calculation effort. In this paper we propose the use of Hadamard-based Sum of Absolute Transformed Differences (SATD), in replacement of the traditionally used Sum of Absolute Differences (SAD), as a means of improving the efficiency of video coding. To allow that we explore two Hadamard Transform methods to design efficient SATD architectures, one using the Fast Hadamard Transform (FHT) butterfly and another one using the so-called Transform-Exempted (TE) SATD algorithm. Those methods were combined with architectural decisions (full parallelism, full parallelism with pipelining or multi-cycling) to build a total of six Hadamard-based SATD architectures that were synthesized for a commercial 45nm standard cell library for two operating frequencies. The architectures were simulated with pixel block data to obtain realistic dynamic power and energy estimates. The TE-SATD architectures achieved the lowest energy results: down to 13.13 pJ/SATD in the case of parallel architecture with pipeline. However, considering also the area results when evaluating energy, the best results are given by both methods using multi-cycling (transpose buffer): nearly 20.75 pJ/SATD with up to 63.54% smaller area compared with fully parallel architectures.

Index Terms: Video Coding; VLSI Design; Sum of Absolute Transformed Differences; Hadamard; Energy Efficiency.

I. INTRODUCTION

In current video coding (VC) standards, inter frame and intra frame prediction techniques are the main responsible for the high compression rates achieved. The former technique explores temporal redundancies between frames, being one of the most time demanding video coding tasks [1]. The latter technique explores spatial redundancies of the already coded blocks from the current frame. A block is a sub-division of the frame (i.e., a sub-matrix of pixels) which is used to facilitate the various coding tasks. The allowed sizes for the blocks (e.g., 16×8, 8×16, 8×8, 4×4) are defined within each video coding standard. In the video coder tool-chain of Fig. 1 it is possible to observe that after selecting a block that best matches with the one being coded (considering both inter and intra frame prediction), the coder computes the residues (D), which are the differences pixel-to-pixel between two similar blocks.

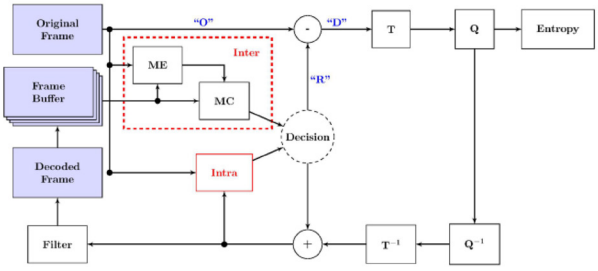


Figure 1. Main tools of the H.264/AVC [4] flow. Inter prediction is presented by the Motion Estimation (ME) and Motion Compensation (MC) blocks. The transform and quantization steps are respectively shown by T and Q blocks.

The residues are then transformed (T) and quantized (Q). The block selection referred to in the previous paragraph is performed by the Block Matching Algorithm (BMA) [2]. Basically, its task can be stated as follows: for each $M \times N$ sized block being coded ($O_{M \times N}$) choose, among the several available candidate

blocks ($C_{M\times N}$), the one that minimizes the similarity metric to use as reference ($R_{M\times N}$). Due to its simplicity, the most used similarity metric is the Sum of Absolute Differences (SAD) [3]. It is calculated as:

$$\mathrm{SAD}_{M\times N} = \sum_{i=1}^{M}\sum_{j=1}^{N} |d_{ij}| \quad (1)$$

where $d_{ij} \in D'_{M\times N} = O_{M\times N} - C_{M\times N}$.

Although for most applications SAD provides an acceptable coding efficiency, more elaborated metrics are required to help improving compression efficiency for current and next generation high definition video standards. Unfortunately, dedicated hardware architectures to implement such metrics are more complex than those generally used to calculate the SAD, resulting in silicon area and power consumption overheads. Therefore, it is essential to optimize their computation so as to reduce as much as possible those overheads, thus making viable their adoption. Aside the SAD, the most well-known similarity metrics are the Sum of Squared Differences (SSD) and the Sum of Absolute Transformed Differences (SATD) [3]. In SSD the square operation is employed to give more weight for larger differences than for smaller ones, which results in coding efficiency increase. Area and power consumption overheads come from the square operation. SATD, by its turn, is able to provide even higher coding efficiency because it correlates better with the transform step, which will be explained in the sequel.

The SATD is defined as:

$$\mathrm{SATD}_{M\times N} = c \times \sum_{i=1}^{M}\sum_{j=1}^{N} |td_{ij}| \quad (2)$$

where $c \in R^{*}_{+}$ is a scaling constant and $td_{ij} \in TD$ is a 2-D transformed differences block, calculated by:

$$\mathrm{TD}_{n\times n} = \mathrm{T}_{n\times n} \times \mathrm{D}'_{n\times n} \times \mathrm{T}^{T}_{n\times n} \quad (3)$$

In Equation 3, T represents the transformation matrix of an integer linear transform. In some works, such as [5], the transformation matrix of the integer Discrete Cosine Transform (DCT) is used. However, it is more likely to find the Walsh-Hadamard Transform (or Hadamard Transform, for shortly) matrix, as in the H.264/AVC reference software (JM) [6]. The recursive definition of Hadamard matrix (H), with size $2_n \times 2_n$ such that $\forall n, \exists k \geq 0 \mid n = 2^k$, is:

$$\mathrm{H}_{2n\times 2n} = \mathrm{H}_{2\times 2} \otimes \mathrm{H}_{n\times n} \quad (4)$$

where $\otimes$ is the Kronecker product and $\mathrm{H}_{2\times 2}$ is the lowest order Hadamard matrix [7], given by:

$$\mathrm{H}_{2\times 2} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (5)$$

Fig. 2 illustrates how to obtain the Hadamard matrix $\mathrm{H}_{8\times 8}$ by using the given recursive definition.

The benefits of using Hadamard-based SATD comes from its correlation with the DCT, the most commonly used transform in step T. Such correlation allows reducing the quantization (Q) error. As it can be seen in Fig. 1, the transform (T) is applied on the residue matrix (D), which is calculated by:

$$\mathrm{D}_{M\times N} = \mathrm{O}_{M\times N} - \mathrm{R}_{M\times N} \quad (6)$$

where (O) is the block being coded and (R) is a reference block chosen by the BMA (using a given similarity metric) from either the intra or the inter prediction. The coefficients resulting from the transform are then quantized (Q). This way, the use of a similarity metric that emulates the behavior of the transform step increases the correlation between prediction and its effect on quantization, thus reducing the generated error.

In short, the Hadamard-based SATD differences matrix (*D*') calculation, 2) the 2-D transformed differences block computation requires the following three steps: 1) the (TD) calculation and 3) the sum expressed in Equation 2.

For 4×4 pixel blocks, which is the block size of interest for this work, the differences matrix $\mathrm{D}'_{4\times 4}$ is obtained through the subtraction between the original

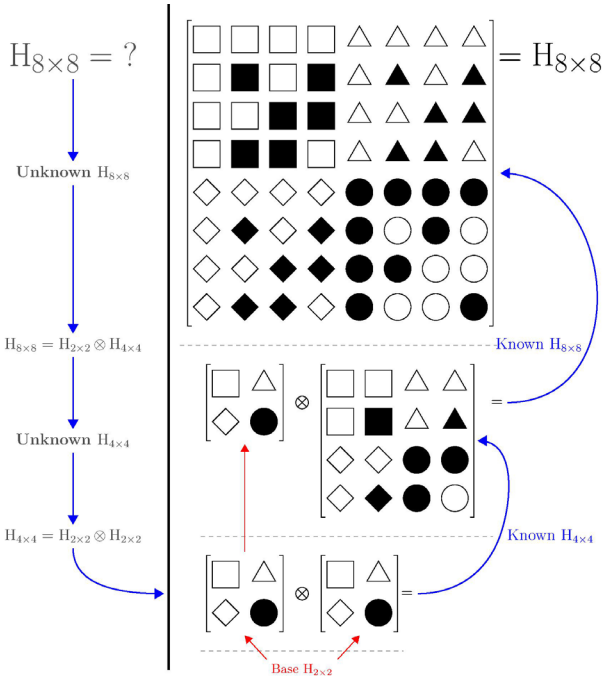


Figure 2. Recursive calculation of the H matrix for n = 8. Filled shapes represents −1, while blank ones represents 1. Notice that the shapes are kept according to Kronecker product (⊗).

block ($O_{4\times4}$) and a candidate block ($C_{4\times4}$). The 2-D transformed differences block (TD) is calculated by using Equation 3, assuming as T the 4 × 4 Hadamard matrix, which is defined as:

$$H_{4\times4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \quad (7)$$

Finally, the sum of Equation 2 becomes:

$$SATD_{4\times4} = \frac{1}{2}\sum_{i=1}^{4}\sum_{j=1}^{4}|td_{ij}| \quad (8)$$

As will be shown after Theorem 1, the 1/2 scaling factor does not reduce the SATD precision because the less significant bit of the resulting value (without scaling) will always be zero.

As expressed in Equation 3, the considered 4 × 4 SATD requires a 2-D Hadamard transform on a residue matrix, which is computed by two matrix multiplications. Those matrix multiplications are the main responsible for the area and power consumption overhead with respect to both SAD and SSD. On the other hand, both quality and compression rates resulting from using SATD are better than those resulting from SAD and SSD [8], [9] because, considering 4×4 blocks, two among four base functions of H4×4 (Equation 7) are the same as for the integer DCT4×4 used in H.264/AVC [4], [10]. Therefore, the SATD is well accepted for intra prediction [11], [12], once the number of candidate blocks is far less than that for inter prediction. For instance, there are only nine 4 × 4 intra candidates in H.264/AVC [4], [13]. However, in HEVC [14] there are 34 candidates in intra 4 × 4 [15]. Such increase in the number of candidates, even for intra prediction, along with the larger number of blocks caused by the adoption of larger resolutions (e.g. 2K and 4K), encourages the adoption of more efficient similarity metrics.

In [16], the authors explore the Hadamard separability property, using two RAMs to store intermediary transform results. The lines and columns are transposed by a so-called "ping-pong" buffer, emulating a transpose buffer as defined in [17]. Also in [16], the pipeline depth and its positioning are explored to build fully parallel Hadamard transform architectures. Unlike these, some works on SATD do not apply the separability property of the Hadamard Transform. This is the case of [9], where the authors use 2-D transform blocks. To the best of our knowledge, none of the publications on the TE-SATD method present hardware architectures, while [10] simply evaluated the number of saved operations in relation to the MMB and FHT methods. In our previous work [18] we analyzed the area and energy of some Hadamard-based SATD architectures.

In this paper, we explore two methods to calculate the 4×4 Hadamard Transform, aiming the design of efficient SATD dedicated architectures. One method is based on the Fast Hadamard Transform (FHT) whereas the other one combines the FHT itself with the Transform Exempted (TE) method [10], [19]. The main contribution of this paper is a thorough evaluation of six SATD architectures that were designed by combining the two Hadamard Transform methods with three micro-architecture options (fully parallel, fully parallel with pipelining or multi-cycling with a transpose buffer). Each of these six architectures were synthesized for a 45nm commercial standard cell library for two target frequencies, totalizing twelve versions of architectures. Those twelve versions were evaluated and compared using the following figures of merit: dynamic and leakage power, energy and silicon area.

The main improvements of this paper are the power and energy analysis by considering more accurate switching activities, obtained through simulation using realistic stimuli.

This paper is organized as follows. Section II details the FHT-based and TE-SATD methods. The six designed architectures are presented in Section III. Section IV presents the synthesis and simulation setup as well as area, power and energy estimates. It also establishes comparisons between twelve versions of architectures. Conclusions are drawn in Section V.

II. HADAMARD-BASED SATD

To achieve efficient hardware architectures for the Hadamard-Based SATD one must, first of all, optimize the calculation method itself. Then, design decisions must be taken so as to take benefit of the particular features of the optimized calculation method.

Among the three steps required to compute the HadamardBased SATD, the second one (the 2-D transformed differences block calculation) is the most complex and also the responsible for the area and power consumption overhead with respect to SAD. Therefore, special attention must be given to its realization.

Two matrix multiplications are needed to calculate the transformed differences matrix of order n, $TD_{n\times n}$ (Equation 3). A naïve method to accomplish this would be to use the ordinary matrix multiplication algorithm, which is $O(n^3)$. Nevertheless n is always a small number, the number of multiplications and additions required is large (e.g., for $n = 4$, $2n^3 = 128$ multiplications and $2n^3 - 2n^2 = 96$ additions). On the other hand, since the H matrix is composed only of $+1$ and -1, a straightforward solution to minimize the number of operations relies on computing the trans-

form using only additions and subtractions. In such case, for $n = 4$ only 96 additions are needed. Finally, the last step (i.e., the sum in Equation 2) is performed by n^2 absolute operations and n^2-1 additions. For $n = 4$ the second and third steps of SATD require 16 absolute operations and 111 additions. Hereinafter, we will refer to this method as Matrix Multiplication-Based SATD, or MMB-SATD for short.

A more efficient method to compute the second step relies on using the so-called Fast Hadamard Transform (FHT) [20]. Basically, it breaks recursively a Hadamard matrix $H_{n\times n}$ into two smaller matrices $H_{n/2\times n/2}$, resulting in a

complexity of $O(n^2\log n)$. Such optimization is made possible by the separability property of the Hadamard Transform, which allows to apply two 1-D transforms on the lines and columns of a given differences matrix [16]. Each 1-D FHT operation, for a single column or row, can be represented by a butterfly, as the one in Fig. 3. The butterfly can be interpreted as a map function defined over a given $\vec{X}$ to another $\vec{T}$. Especially in the case of a 4 × 4 Hadamard, it is defined as $\vec{X}(x_1, x_2, x_3, x_4) \mapsto \vec{T}(t_1, t_2, t_3, t_4)$ as follows:

$$\vec{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \mapsto \vec{T} = \begin{bmatrix} (x_1 + x_2) + (x_3 + x_4) \\ (x_1 - x_2) + (x_3 - x_4) \\ (x_1 + x_2) - (x_3 + x_4) \\ (x_1 - x_2) - (x_3 - x_4) \end{bmatrix} \quad (9)$$

Each butterfly requires $n\log n$ additions and is executed 2^n times, resulting in a total of $2n^2\log n$ additions. For n = 4, 64 additions are needed which is 33% less than in the MMB-SATD method. That is why the FHT is the method of choice to compute the Hadamard transform in many works (e.g. [21], [17], [16]), including the Hadamard-based SATD [9]. As in the MMB-SATD method, the last step requires n 2 absolute operations and $n^2 - 1$ additions. Hence, for $n = 4$ the second and third steps of the FHT-SATD requires 16 absolute operations and 79 additions, which is nearly 25% less additions than the MMB-SATD method.

The third method considered in this paper is called Transform-Exempted (TE) SATD [10], [19]. By using the FHT, it combines the second and third steps of the Hadamard-Based SATD by absorbing part of the absolute sum into the second 1-D butterfly step of the FHT. By doing so, it achieves 38% and 17% of operation reductions for computing a 4×4 SATD in comparison to MMB and FHT-based methods, respectively. Such algorithm relies on the following property, proved in [10]:

Property 1.

$$|a + b| + |a - b| = x \times max(|a|, |b|) \quad (10)$$

To demonstrate the TE-SATD, Equation 9 can be re-written as:

$$\vec{T} = \begin{bmatrix} (x_1 + x_2) + (x_3 + x_4) \\ (x_1 - x_2) + (x_3 - x_4) \\ (x_1 + x_2) - (x_3 + x_4) \\ (x_1 - x_2) - (x_3 - x_4) \end{bmatrix} = \begin{bmatrix} y_1 + y_2 \\ y_3 + y_4 \\ y_1 - y_2 \\ y_3 - y_4 \end{bmatrix} \quad (11)$$

As mentioned before, $\vec{T}$ is a 1-D FHT transformed vector. Hence, it can be seen as the result of the second 1-D transform. By such definition, let every t_{ij} be one element $t_j \in \vec{T}^T$ from a line i of TD. Thereby, replacing t_j by its form in the last factor of Equation 11, Equation 8 can also be re-written as:

$$\text{SATD}_{4\times4} = \frac{1}{2}\sum_{i=1}^{4}\sum_{j=1}^{4}|td_{ij}| \quad (12)$$

$$= \frac{1}{2}\sum_{i=1}^{4}(|y_1 + y_2| + |y_1 - y_2| + |y_3 + y_4| + |y_3 - y_4|)_i$$

By applying Property 1 to Equation 12, the following holds:

$$\text{SATD}_{4\times4} = \sum_{i=1}^{4}(max(|y_1|, |y_2|) + \max(|y_3|, |y_4|))_i \quad (13)$$

Therefore, instead of calculating the second butterfly (which requires $n^2\log_2 n$ additions) and $n^2 - 1$ additions for the third step, the TE-SATD method requires only $n^2 + n$ additions, $n^2/4$ maximum values and n additions to perform the addition within the parenthesis in Equation 13 and $n^2/4 - 1$ additions for the third step. For $n = 4$, TE-SATD requires only 55 additions, 8 maximum values and 16 absolute operations.

Table I summarizes the number of operations for the three presented SATD methods. Despite the absolute operation takes a similar time than the addition, the former is not considered in Table I since all three methods require the same number of absolute operations. For the same reason, Table I does not include the subtractions required in the first step of the three methods.

Among the three previously described methods, the FHT and TE are the ones requiring the least amount of operations. For such reason, they were selected as the starting point to investigate energy-efficient SATD architectural implementations. The main difference between those two methods relies on the second 1-D transform step and on the sum depth.

Table I. Number of Operations for SATD algorithms – Additions (A) and Maximum Values (M)

Method	*n*	*n* = 4
MMB-SATD	$2n^3 - n^2 - 1A$	111A
FHT-SATD	$(2\log_2 n+1)n^2 - 1A$	79A
TE-SATD	$(\log_2 n + 5/4)n^2 + n - 1A + (n^2/2)M$	55A + 8M

Fig. 4 shows the designed datapath for the second step of FHT. The T block represents a 1-D FHT butterfly, and performs the operations shown in Equation 9 (Fig. 3). The shift right operations are consequence of the following theorem:

Theorem 1. *All elements of* $\vec{T}$ *have the same parity.*
$\forall t_i, t_j \in \vec{T}$, t_i *is even* $\Leftrightarrow$ t_j *is even*

Proof: Let every $t_j \in \vec{T}$ be written as $(x1 \pm x2) \pm (x3 \pm x4)$ (Equation 9). Since the parity is the same for addition as for subtraction, it follows that the parity of all $t_j \in \vec{T}$ are the same. ■

Since all transformed values have the same parity, the result of adding two of them together is always even i.e., the least significant bit (LSB) is 0. Therefore, no loss is introduced if those values are divided by two after they are added together. In this case, the final division by two (constant $c = 1/2$ in Equation 8) is performed earlier, while maintaining the originally defined range.

The datapath for the TE-SATD algorithm is shown in Fig. 5. Basically it operates as defined in Property 1, except that the multiplication by two is not performed because it was canceled by the scaling factor in SATD definition (see Equation 13). It is also possible to observe the absence of the T block and the reordering of the TE-SATD inputs, reflecting the organization presented in Equation 12.

The two presented datapaths were combined with each one of the three base architectures described in the next section, giving rise to six Hadamard-based SATD architectures.

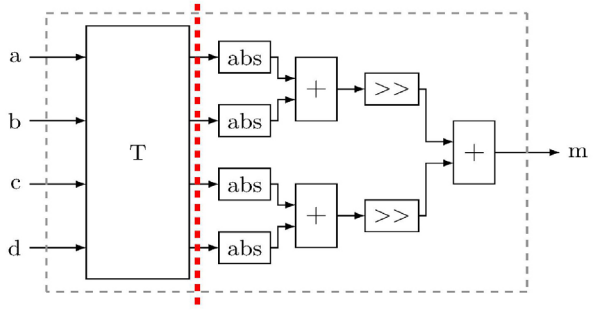


Figure 4. Fig. 4 Fast Hadamard Transform (FHT) method block (M) datapath for the second 1-D FHT along with the sum of absolute transformed values of one row or column.

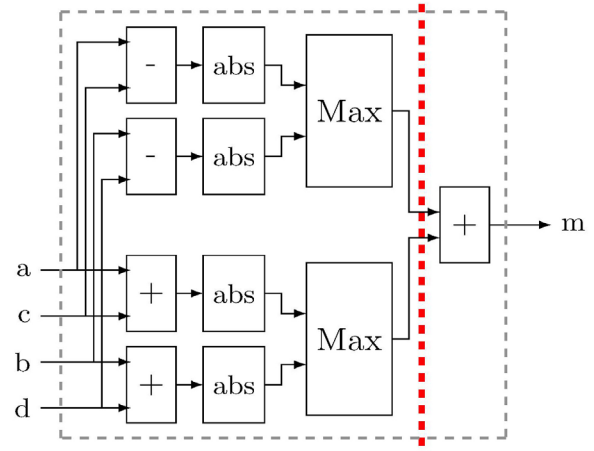


Figure 5. Transform-Exempted (TE method block (M) datapath which takes four inputs already transformed by one 1-D FHT

III. BASE ARCHITECTURES

Half of the six Hadamard-based 4×4 SATD architectures are based on the Fast Hadamard Transform (FHT) whereas the other half use a combination of the FHT with the Transform Exempted (TE) method. Unlike most correlated works, power and energy assessments assume a realistic framework in which the architectures were simulated with input stimuli generated from a real HD video sequence.

Although the HEVC standard defines Prediction Units (PUs) that are larger than 4×4, many VC works use small block sizes. Thus, we believe that the design and evaluation of 4×4 SATD is still a valid concern. A recent application example supporting our claim is the Fast Intra Prediction Algorithm proposed by [22], where 4×4 and 2×2 block**.** based Hadamard transforms are used instead of the original 8×8 and 4×4 based transforms of the HEVC reference software (HM) [23]. In this section we present the base architectures to compute the three steps in both FHT-SATD and TE-SATD.

The so-called "Tree" architecture, shown in Fig. 6, is fully parallel. As first step, it performs 16 subtractions between two 4 × 4 pixel blocks and then

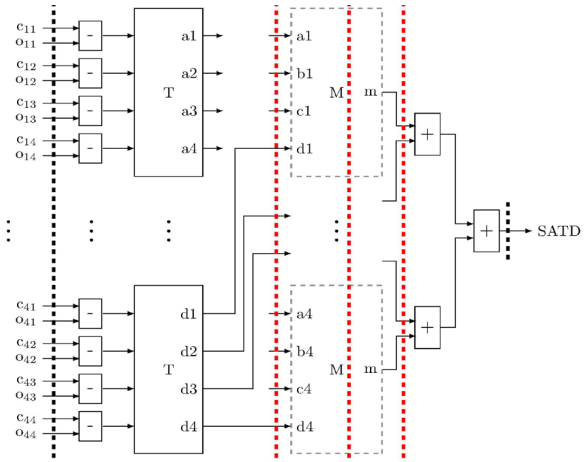
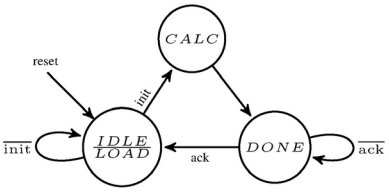


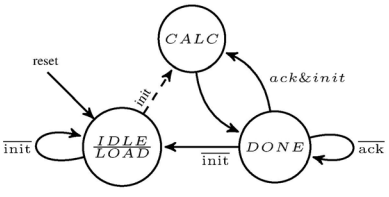
Figure 6. Datapath of Tree and Tree-4stages architectures. The M blocks represent the method blocks, which can be either the FHT block presented in Fig. 4 or the TE block presented in Fig. 5. The straight dashed lines represent registers: the Tree architecture has only input and output registers, whereas the Tree-4stages architecture has pipeline registers (intermediate straight dashed lines).

performs in parallel four 1-D FHT (T blocks) on the resulting differences. Hence, the connections between the T blocks and the method blocks (M blocks in Fig. 6) implement the matrix transposition. The next step is performed by the method block, which can be either the FHT (Fig. 4) or the TE (Fig. 5). The remaining sums of the third step of the SATD calculation is performed by summing together all four transformed lines (outputs marked as "m" in the method blocks). Such parallel base architecture allows to perform the 4×4 SATD calculation in a single clock cycle. However, it is important to notice that the Tree architecture requires a large adder tree, resulting in a long critical path. The Finite State Machine (FSM) to control the Tree architecture has three states as presented in Fig. 7a. The SATD computation takes place in state CALC, whereas the other two states are used for synchronization.

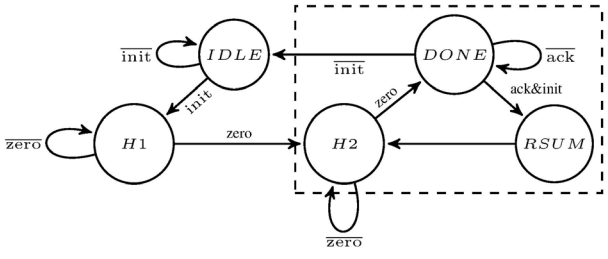
The "Tree-4stages" architecture was designed looking for reducing the clock cycle through the addition of pipeline registers to the Tree architecture, as represented by the dashed vertical lines in Fig. 6. It is interesting to observe that even the four method blocks are crossed by a pipeline register. The pipeline register positions were fixed prior to synthesis. The control of the Tree-4stages architecture uses a similar FSM (Fig. 7b) as the one used by the Tree architecture, except that for each pipeline stage the pair CALC–DONE is executed.



(a) Tree FSM.



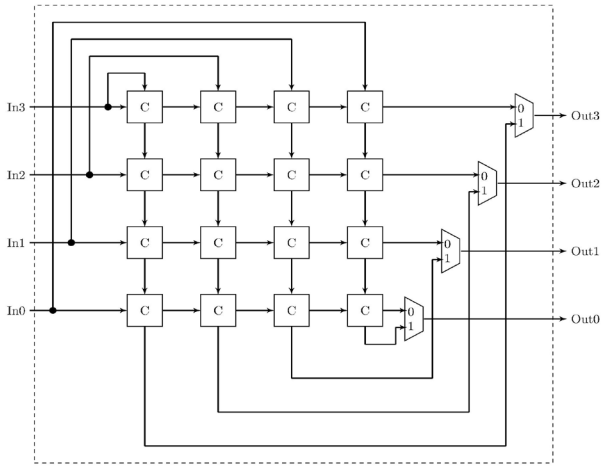
(b) Tree-4stages FSM (Excerpt).



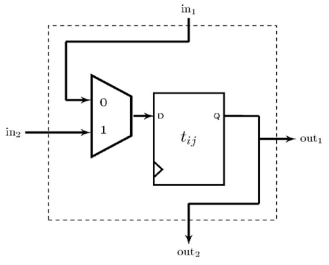
(c) Transpose FSM (Excerpt).

Figure 7. Finite State Machines (FSMs) used for controlling the base architectures.

The third architecture, called "Transpose", adopts the transpose buffer (TB) shown in Fig. 8, which is similar to the ones presented in [17] and [24]. The Transpose architecture utilizes only one T block and one method block, connected together by the TB, as shown in Fig. 9. It also uses four subtractors to generate the differences matrix and one accumulator to perform the final sum. The Transpose FSM is presented in Fig. 7c. In state H1, the transpose buffer is fed line by line with the 1-D transformed



(a) Transpose Buffer (TB).



(b) TB Cell (C).

Figure 8. Transpose Buffer (TB) design and its storage cells (C). The writing/reading direction of the transpose buffer is controlled by the MSBof one 3-bit counter.

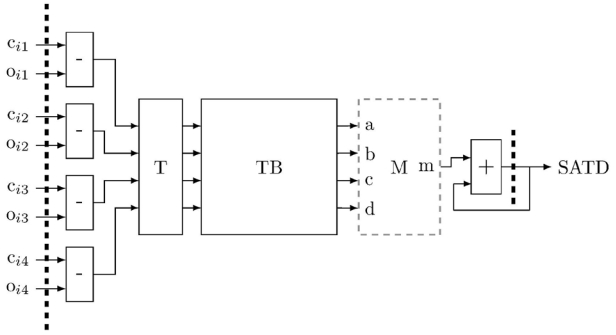


Figure 9. Datapath of Transposed 2T architecture. Input and output registers are represented by straight dashed lines. The M block is the method block. The T block is a 1-D FHT transform and performs the operations shown in Fig. 3. The transpose buffer (TB) is detailed in Fig. 8.

differences. After four clock cycles in state H1, the FSM moves to H2 state. In H2 the TB dispatches the received results to the method block column by column during the next four cycles, while receives new arriving data column by column. After that, the FSM moves to DONE state and the first SATD is computed. Its worth noting that this architecture has a latency of eight clock cycles: four cycles in H1 state and four cycles in H2 state. When ready to proceed, the computed SATD is cleared in state RSUM. From this point on, a new SATD is computed after every four clock cycles in state H2.

IV. RESULTS

Six Hadamard-based SATD architectures were devised by replacing the M blocks in each of the three architectures detailed in Section III (Tree, Tree-4stages and Transpose) by either FHT-SATD or TE-SATD methods presented in Section II. The six architectures were then described in Verilog and synthesized with Synopsys R Design Compiler (DC R) [25] in Topographical mode for nominal voltage (0.7V) 45 nm TSMC [26] standard cell library for two operation frequencies, one for target throughput and another for maximum, resulting in a total of 12 cases. The target throughput is the same one as assumed in [27] for SAD: 16 million 4×4 blocks/s which, according to the authors, is required for coding a 1080p video sample at 30 fps.

Table II lists the maximum achieved frequencies and the required target frequencies for all synthesis cases. The listed maximum frequencies are the maximum ones for which the synthesis tool could meet the timing constraints. However, notice that reaching such high frequencies only allows for evaluating the maximum throughput allowed by the architecture itself when combined to the used standard cell library, which is dependent on the size and number of critical paths (architecture) and the possible cells to use in

such paths (library). Furthermore, such high frequencies can be required in the case of using larger spatial or temporal resolutions (larger frame sizes and higher number of frames per second, respectively).

Table II. Maximun (Max) and target (Tar) frequencies (MHz)

	FHT			TE		
	Tree	Tree-4	Transpose	Tree	Tree-4	Transpose
Max	778.12	2064.52	1273.63	842.1	1954.2	1248.78
Tar	48.08	32.00	95.97	48.08	32.0	95.97

Because our architectures were designed isolated from any surrounding BMA architecture, there is no insurance on the devices that will drive the SATD architectures' inputs, nor the ones that will be driven by our architectures. Therefore, one good practice is to create both time and load budgets [8]. Concerning time, we took into account the latest data arrival time in all devised architectures' inputs by conservatively limit the input delay to 60% of the clock period. The same constraining were done for the outputs of our architectures, by limiting the output delays also to 60% of the clock period. Both input and output delay constraints specify how much time is used by external logic and DC calculates how much time is left for the internal logic [28]. Concerning load, the maximum primary input capacitance was set to 10 times a 2-input AND gate whereas the maximum primary output capacitance was set to 30 times a 2-input AND gate. Both the time and load budgets are based on [28].

Area and static power estimates were obtained directly from synthesis. A testbench was written in Verilog to validate the architecture and also to obtain the switching activity file (SAIF) for each synthesized netlist. Prior to simulation, we have run a modified version of the x264 encoder with full HD sequence "Pedestrian Area"[29] thus dumping its pixel block data. To limit simulation runtime we only simulated 1 million 4 × 4 SATD calculations for each synthesis case. One specific handler for each architecture was programmed in C to interface between the pixel data file and the Verilog testbench. The SAIF generated after simulation with Synopsys VCS® [30] was further used to obtain realistic dynamic power estimates.

Fig.10 shows the relative distribution of static and dynamic power on total power results, along with the absolute total power figures. The impact of dynamic power is larger for maximum throughputs,

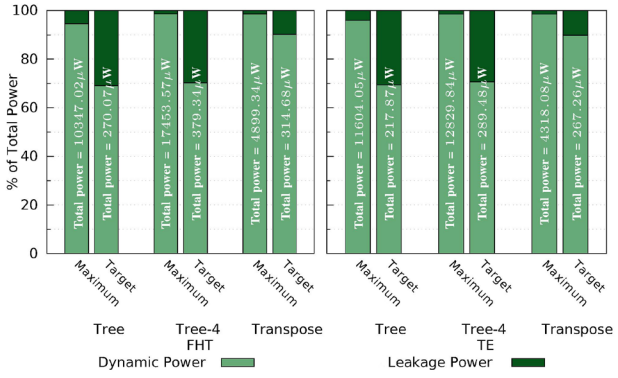


Figure 10. Relative distribution of static and dynamic power on total power results, for which absolute values are presented for each synthesis case.

while the static power is more prominent in both Tree and Tree-4stages architectures for target throughput. Fig.11 presents total power for comparison purposes, showing the large total power differences between maximum and target throughput syntheses.

By using the obtained total power results, we estimate the energy/SATD, after Equation 14, where E_{SATD} is the energy for each SATD calculation, Ptot is the total power, C is the total number of cycles for each SATD and T is the clock period. Notice that $C \times T$ is the time spent to perform one SATD calculation. Hence, for the same throughput, $C \times T$ is constant.

$$E_{SATD} = P_{tot} \times C \times T \tag{14}$$

Fig. 12 presents energy/SATD estimates for the 12 synthesis cases. Let us first analyze the results for the target throughput. The Tree architecture with TE method presented approximately 19.2% less energy consumption when compared to the Tree architecture with FHT. Once both architectures operate to achieve the same throughput, power is the only influencing factor, which is lower in TE-SATD due to its shorter critical path. A shorter critical path is a key factor for reducing power in Tree architecture, given its large number of critical paths to be optimized.

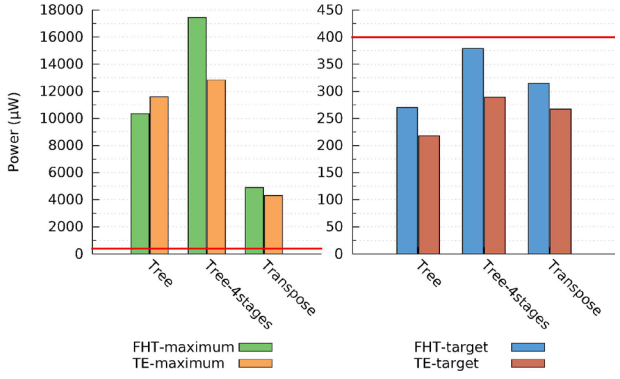


Figure 11. Absolute power figures. To allow the visualization of differences between target throughput synthesis cases, we show them in a different scale. The solid horizontal line shows the equivalence between scales.

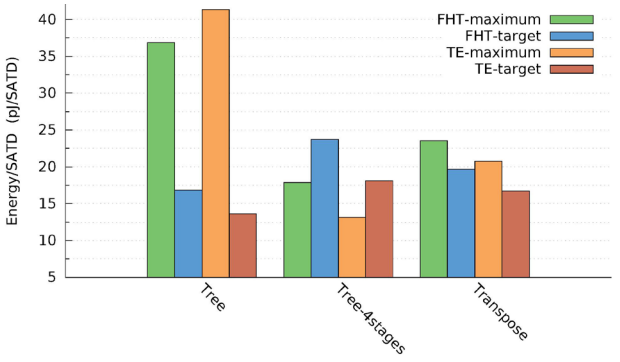


Figure 12. Energy estimates for each architecture. The marks A, B and C refers to three kinds of comparisons performed, respectively: method, architecture and throughput.

In Tree-4stages architecture the energy differences between the two methods is slightly more prominent: TE method demanded 23.7% less energy than FHT. A possible explanation to such behavior would be the positioning chosen for the pipeline registers, which might have led to excessively long critical path in FHT, resulting in the selection of faster cells and consequently, increasing power. A viable workaround is to allow the synthesis tool to perform retiming [31], when DC® moves registers within combinational logic gates to optimize time and area [32]. As for Transpose architecture, TE method presented 15% energy reduction when compared to FHT. For such architecture the transpose buffer plays a major role in balancing the energy consumption along the datapath since, according to [33], the transpose buffer can represent a significant part of the power consumed by a 2D transform.

Still concerning target throughput, Tree-4stages architecture presented higher energy consumption when comparing the same method. The inclusion of pipeline registers compromised both static and dynamic power, resulting in a higher total power, as can be seen in Fig.11.

Even presenting the same relative static and dynamic power distribution as Tree-4stages, Tree architecture required less total power, resulting in the lowest energy.

Let us now analyze the results for maximum throughput. To compare these cases with each other, one must remind that the energy variations (calculated after Eq.14) are now influenced by both power and SATD calculation time ($C \times T$). Therefore, all frequencies ($1/T$) are presented in Tab.II. Unlike all other synthesis cases, for Tree architecture the FHT method presented 10.83% less energy than TE. However, its larger number of long paths led to more optimizations to meet timing constraints, requiring faster and thus, more power demanding cells. Combined with the small achieved maximum throughputs (for both FHT and TE methods) such optimizations led also to high energy consumptions: up to 1.56 times more than any other architecture.

On the other hand, Tree-4stages presented the lowest energy figures. By applying TE method, such architecture requires almost 25% less energy than by applying FHT. Once again, the TB plays a major role in balancing both methods: Transpose presented the closest energy estimates between TE and FHT methods. The former required 11.85% less energy than the latter.

Thanks to both pipeline and parallelism, Tree-4stages achieved the highest throughputs. As counterpart, the achieved throughputs led to the lowest energy figures among the maximum throughput cases, even with power results up to 3.56 times higher than those of the Transpose architecture.

Considering both target and maximum throughput syntheses, the most energy-efficient case is the Tree architecture with TE method synthesized for target frequency. On the other hand, when synthesized for maximum throughput, Tree architecture presents the worst energy figures. Considering the cases using TE method, Transpose architecture requires 1.22 times more energy than Tree at target frequency, whilst at maximum frequency it consumes near 50% less. By its turn, Tree-4stages requires more energy than any other architecture for target frequency, whereas presenting results that are very close to those of Transpose combined with the TE method.

Both Tree and Transpose based architectures present lower energy consumption for target throughput than for maximum. However, Tree-4stages based architecture presents lower energy for maximum throughput syntheses. Basically, the short critical paths allowed higher frequencies, thus reducing the impact of the higher power values on the resulting energy.

Fig. 13 shows the area estimates for the six designed architectures. All the maximum throughput syntheses presented larger areas than their target counterparts. Such was due tighter timing constraints require faster gates which, by their turn, are also larger. Such increase, in opposition to energy behavior, cannot be compensated by other factors. Considering target throughput syntheses, it becomes clear the effects of parallelism and pipeline: the less parallel architecture (Transpose) presented smaller area, whilst the one with most parallelism and pipeline (Tree-4stages) presented larger areas.

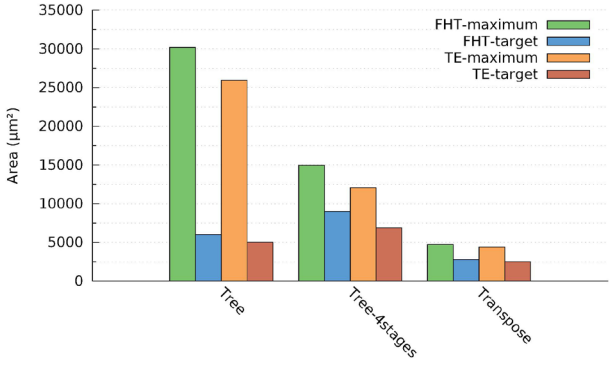


Figure 13. Area results for all synthesis cases.

V. CONCLUSIONS

Despite the optimizations applied to the FHT, such as early parity detection (based on Theorem 1), TE-SATD still presents itself as the best alternative to implement dedicated hardware to calculate the Hadamard-based SATD. By merging the last level of the transform (in the second step) with the first absolute sums (in the third step), the TE-SATD method can reduce both area and energy consumption. The only case in which FHT presented better energy estimates than TE was the Tree architecture for maximum frequency. On the other hand, the TE-SATD method reveals its greatest advantage on Tree- 4stages synthesized for maximum throughput, by consuming 26.5% less energy and occupying 19.25% less area than FHT. In addition, such case presented the lowest energy/SATD: 13.13 pJ/SATD. However, its area was 2.74 times larger than that the required by the Transpose architectures which, by their turn, resulted in nearly 20.75 pJ/SATD of energy consumption.

Finally, the best trade-off between energy consumption and area is shown by Transpose architecture. Also, unlike the results obtained for Tree and Tree-4stages, those obtained for Transpose were very similar for all methods, with TE featuring an area reduction of 9.8% and 15.05% less energy per block when compared to FHT. Such convergence between the two methods when using the transpose architecture is due to two reasons: less parallelism (fewer instances of the method block) and dominance of the transpose buffer on the overall required logic.

ACKNOWLEDGEMENTS

This work was partially supported by the Brazilian Federal Agency for the Support and Evaluation of Graduate Education (CAPES) and by the Brazilian Council for Scientific and Technological Development (CNPq).

REFERENCES

[1] F. Bossen, B. Bross, K. Suhring, and D. Flynn, “Hevc complexity and implementation analysis,” IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1685–1696, 2012.

[2] I. Chakrabarti, K. Batta, and S. Chatterjee, Motion Estimation for Video Coding: Efficient Algorithms and Architectures, ser. Studies in Computational Intelligence. Springer International Publishing, 2015.

[3] I. E. Richardson, The H.264 Advanced Video Compression Standard, Second Edition. John Wiley & Sons Ltd, 2010.

[4] ITU-T, “H.264 corrigendum 1,” jan 2009.

[5] Y.-W. Huang, B.-Y. Hsieh, T.-C. Chen, and L.-G. Chen, "Analysis, fast algorithm, and vlsi architecture design for h.264/avc intra frame coder," IEEE Trans. Circuits Syst. Video Technol., vol. 15, no. 3, pp. 378–401, March 2005.

[6] JVT, "Jm joint video team reference software," 2011. [Online]. Available: http://iphome.hhi.de/suehring/tml/

[7] W. Pratt, J. Kane, and H. C. Andrews, "Hadamard transform image coding," Proceedings of the IEEE, vol. 57, no. 1, pp. 58–68, Jan 1969.

[8] L.-M. Po and K. Guo, "Transform-domain fast sum of the squared difference computation for h.264/avc rate-distortion optimization," IEEE Trans. Circuits Syst. Video Technol., vol. 17, no. 6, pp. 765–773, 2007.

[9] J. S. Dominges Jr, V. N. Possani, D. S. Silveira, L. S. da Rosa Jr, and L. V. Agostini, "High throughput 4×4 and 8×8 satd similarity criteria architectures for video coding applications," in 2011 VII Designer Forum (DF), 2011, p. 115.

[10] C. Zhu and B. Xiong, "Transform-exempted calculation of sum of absolute hadamard transformed differences," IEEE Trans. Circuits Syst. Video Technol., vol. 19, no. 8, pp. 1183–1188, 2009.

[11] H. M. Wang, C.-H. Tseng, and J. F. Yang, "Computation reduction for intra 4×4 mode decision with satd criterion in h.264/avc," Signal Processing, IET, vol. 1, no. 3, pp. 121–127, September 2007.

[12] J. Kim and J. Jeong, "Fast intra mode decision algorithm using the sum of absolute transformed differences," in Digital Image Computing Techniques and Applications (DICTA), 2011 International Conference on, Dec 2011, pp. 655–659.

[13] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 560–576, July 2003.

[14] ITU-T, "Recommendation itu-t h.265: High efficiency video coding," International Telecommunication Union, Geneva, Recommendation H.265, 2013.

[15] G. Sullivan, J. 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, dez 2012.

[16] M. Porto, T. da Silva, R. Porto, L. Agostini, I. da Silva, and S. Bampi, "Design space exploration on the H.264 4×4 hadamard transform," in NORCHIP Conference, 2005. 23rd, Nov 2005, pp. 188–191.

[17] T.-C. Wang, Y.-W. Huang, H.-C. Fang, and L.-G. Chen, "Parallel 4×4 2d transform and inverse transform architecture for mpeg-4 avc/h.264," in Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on, vol. 2, May 2003, pp. II–800–II–803 vol.2.

[18] L. H. Cancellier, A. B. Bräscher, I. Seidel, and J. L. Güntzel, "Energyefficient hadamard-based satd architectures," in Proceedings of the 27th Symposium on Integrated Circuits and Systems Design, ser. SBCCI '14. New York, NY, USA: ACM, 2014, pp. 36:1–36:6.

[19] F. Jou, "Method for fast satd estimation," Sep. 28 2010, US Patent 7,804,900. [Online]. Available: http://www.google.com/patents/US7804900.

[20] C.-P. Fan and J.-F. Yang, "Fast center weighted hadamard transform algorithms," IEEE Trans. Circuits Syst. II, vol. 45, no. 3, pp. 429–432, Mar 1998.

[21] ——, "Fixed-pipeline two-dimensional hadamard transform algorithms," IEEE Trans. Signal Process., vol. 45, no. 6, pp. 1669–1674, Jun 1997.

[22] H. Zhang and Z. Ma, "Fast intra prediction for high efficiency video coding," in Advances in Multimedia Information Processing – PCM 2012, ser. Lecture Notes in Computer Science, W. Lin, D. Xu, A. Ho, J. Wu, Y. He, J. Cai, M. Kankanhalli, and M.-T. Sun, Eds. Springer Berlin Heidelberg, 2012, vol. 7674, pp. 568–577.

[23] JCT-VC, "Hevc test model," 2013. [Online]. Available: http://hevc.hhi.fraunhofer.de/

[24] V. d. S. Livramento, B. G. Moraes, B. A. Machado, E. Boabaid, and J. L. Güntzel, "Evaluating the impact of architectural decisions on the energy efficiency of fdct/idct configurable ip cores," Journal of Integrated Circuits and Systems, vol. 7, no. 1, pp. 23–36, 2012.

[25] Synopsys, "Synopsys design compiler, version f-2011.09-sp5-2." 2011.

[26] TSMC STANDARD CELL Library TCBN45GSBWPTC, TSMC, 2011.

[27] F. Walter, C. Diniz, and S. Bampi, "Synthesis and comparison of low-power high-throughput architectures for sad calculation," Analog Integrated Circuits and Signal Processing, vol. 73, no. 3, pp. 873–884, 2012.

[28] Synopsys R , "Design Compiler 1 Workshop – Student Guide," Synopsys R Custumer Education Services, Workshop, 2010.

[29] Xiph.org, "Xiph.org test media repository," 2011. [Online]. Available: http://media.xiph.org/

[30] Synopsys, "Synopsys vcs, version g-2012.09." 2012.

[31] N. Shenoy, "Retiming: Theory and practice," Integr. VLSI J., vol. 22, no. 1-2, pp. 1–21, Aug. 1997.

[32] Synopsys, Synopsys's Design Compiler User Guide, Version I-2013.12- SP4., 2014.

[33] R. Rithe, C.-C. Cheng, and A. Chandrakasan, "Quad full-hd transform engine for dual-standard low-power video coding," in Solid State Circuits Conference (A-SSCC), 2011 IEEE Asian, Nov 2011, pp. 401–404.