A HIGH PERFORMANCE AND LOW POWER HARDWARE ARCHITECTURE FOR THE TRANSFORM & QUANTIZATION STAGES IN H.264

Muhsen Owaida, Maria Koziri, Ioannis Katsavounidis, and George Stamoulis

Department of Computer and Communication Engineering,
University of Thessaly, Volos, Greece
(e-mail: {mowaida, mkoziri, ioannis.k, georges}@inf.uth.gr)

ABSTRACT

In this work, we present a hardware architecture prototype for the various types of transforms and the accompanying quantization, supported in H.264 baseline profile video encoding standard. The proposed architecture achieves high performance and can satisfy Quad Full High Definition (QFHD) (3840x2160@150Hz) coding. The transforms are implemented using only add and shift operations, which reduces the computation overhead. A modification in the quantization equations representation is suggested to remove the absolute value and resign operation stages overhead. Additionally, a post-scale Hadamard transform computation is presented. The architecture can achieve a reduction of about 20% in power consumption, compared to existing implementations.

Index Terms— H.264, MPEG-4 AVC, video coding, transform, quantization, low-power, VLSI architecture, hardware implementation

1. INTRODUCTION

Efficient digital video coding techniques are increasingly gaining interest due to the proliferation of low bit-rate video streaming applications (like video-telephony and video-conferencing) but also the worldwide adoption of Digital TV. This increasing interest provided fertile ground for the relevant working groups to develop a new video coding standard, the H.264/MPEG-4 Part 10 standard [1] that would provide the required improved coding efficiency.

H.264, also known as AVC (Advanced Video Coding), was published both by the ISO/IEC and ITU-T [2] and its primary goal is to achieve higher compression ratio while preserving video quality. In order to achieve its goal, H.264 provides superior coding tools, some of which are multiple reference frames, allowing up to 32 reference pictures to be used (unlike prior standards, where the limit was typically one or two) [3], variable block-size motion compensation (VBSMC), enabling very precise segmentation of moving regions [3], [4] in-loop de-blocking filter [3] and context-adaptive binary arithmetic entropy encoding (CABAC) [3].

Furthermore, H.264 introduces an integer orthogonal approximation to the DCT, allowing for bit-exact implementation for all encoders and decoders, without the problem of inverse DCT rounding drift of previous standards [5]. A secondary Hadamard transform is performed on DC coefficients of the primary spatial transform, for chroma DC coefficients and also in one special case for luma DC coefficients, to obtain even more compression in smooth regions. Finally, a scalar power-law quantization is used. The quantizer steps are arranged in a way that there is an increase of approximately 12.5% from one quantization parameter (QP) value to the next [4].

In this paper a hardware architecture prototype for the various types of transforms (4×4 DCT-based, 4×4 and 2×2 Hadamard transforms) supported in the H.264 baseline profile, along with an architecture prototype for the accompanying quantization, are presented.

The rest of the paper is organized as follows: in section 2, the transform and quantization operations, as supported by the H.264 standard, are presented. Section 3 describes the algorithmic modifications made for hardware acceleration of transform and quantization, whereas in section 4 the proposed hardware architectures along with the simulation and experimental results are discussed. Finally, section 5 concludes the paper.

2. TRANSFORM AND QUANTIZATION IN H.264

H.264 supports a hierarchical transform path which includes two levels of different transforms. The first transform level is a 4×4 integer DCT-based transform (also called 'core' transform) that operates on the input 4×4 residual blocks. The transform operation can be written as follows

$$\hat{W} = C_f X C_f^T \tag{1}$$

where the matrix X is the input 4×4 residual block, and C_f is given by the following

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \tag{2}$$

The second level of transform (Hadamard transform) is used in two cases with two different sizes and is applied only on the DC coefficients gathered from each transformed residual block in the first stage.

The first case is that of the chroma components. A 2×2 Hadamard transform is applied on each one of the two 2×2 chroma-DC coefficient blocks, taken by the four 4x4 blocks corresponding to the U and V components, as described below

$$Y_Q = H_1 \hat{W}_Q H_1^T \tag{3}$$

where the matrix H_1 is given by the following

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (4)$$

For the case of a macroblock coded in 16×16 Intra prediction mode (also known as '16Intra'), its luma pixels are first transformed using the 'core' transform described earlier and, as a second step, a gathered 4×4 DC coefficients block is transformed again using a 4×4 Hadamard transform as described by the following set of equations

$$Y = \left(H_2 \hat{W} H_2^T\right)/2 \quad (5)$$

where the matrix H_2 is given by the following

$$H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad (6)$$

It should be noted that luma residual blocks predicted in prediction mode other than 16×16 intra prediction mode are transformed with core transform only.

H.264 utilizes a scalar quantization formula that expresses all scaling factors in fixed point arithmetic, in a way to substitute integer division by an integer multiplication and a shift. This formula can be directly applied to the output coefficients, W_{ij} (for the core transform) and $Y_{D(i,j)}$ (for the Hadamard transform), of the transformation process. The following equations show the quantization formulas used in these two cases.

$$Z_{ij} = round \left(W_{ij} \frac{MF_{ij}}{2^{qbits}} \right) \quad (7)$$

$$Z_{D(i,j)} = round \left(Y_{D(i,j)} \frac{MF_{00}}{2^{qbits+1}} \right) \quad (8)$$

$$qbits = 15 + QP/6 \quad (9)$$

QP is the integer quantization parameter that takes a value in the range [*0–51*] and *MF* is a multiplication factor that takes positive integer values, depending on *QP* and the element's position in the matrix as stated in [2], [3].

In the standard context, equations, (7) and (8) are rewritten in integer arithmetic as shown in equations (10) and (11), respectively.

$$\left|Z_{ij}\right| = \left(\left|W_{ij}\right| . MF + f\right) >> qbits \quad (10)$$
$$sign(Z_{ij}) = sign(W_{ij})$$

$$\left|Z_{D(i,j)}\right| = \left(\left|Y_{D(i,j)}\right| . MF_{(0,0)} + 2f\right) >> (qbits + 1) \quad (11)$$
$$sign(Z_{D(i,j)}) = sign(Y_{D(i,j)})$$

In the above and subsequent formulae, ">>" indicates a binary shift right, and *f* is $(2^{qbits}/3)$ for Intra predicted blocks or $(2^{qbits}/6)$ for Inter predicted blocks.

3. PROPOSED MODIFICATION

Two significant improvements are proposed to achieve high performance architecture. The first concerns the computation of the Hadamard transform, while the second addresses modifications on the integer arithmetic implementation of the quantization equations.

As a first improvement, we propose to compute the Hadamard transform in a cumulative approach after scaling the DC coefficients. According to equations (5) and (6), we can re-write the Hadamard transform as follows, where a_{ri} and a_{jc} are elements of the matrix H_2 in equation (6).

$$Y_{rc} = \left(\sum_{j=0}^{3} \sum_{i=0}^{3} \left(a_{ri} \cdot a_{jc}\right) \cdot w_{ij} \right) \Big/ 2 \quad (12)$$

From equation (12), if we know the index (*i, j*) of the DC coefficient, we can calculate the value of (a_{ri} . a_{jc}) for each Y_{rc}, and then accumulate the DC coefficients over clock cycles. Since the values of a_{ri} and a_{jc} are ±1, only addition or subtraction is performed in the accumulation process.

The quantization equation (11) shows that we need to multiply all the Hadamard transform coefficients by the constant value "MF_{00}". This can be re-written as shown on the left side of equation (13), below. Applying the distributive property of multiplication over addition, we obtain the right side of (13), where we can see that we can multiply each DC coefficient by the multiplication factor before performing the Hadamard transform. This solution may look costly because we need larger sized add/sub units, but if we receive the DC coefficients sequentially from the previous processing units and apply a cumulative approach, only one multiplier is needed for processing up to 16 Hadamard results in total.

$$MF_{00}\left(w_{00} \pm w_{01} \pm \cdots\right) = \left(w_{00} \cdot MF_{00} \pm w_{01} \cdot MF_{00} \pm \cdots\right) \quad (13)$$

The above discussion and results are also correct for the 2×2 Hadamard transform, where we use the Hadamard matrix from equation (4) (with *j, i* ={ 0, 1}).

Care should be taken to properly implement the "division by 2" operation, in case of 4×4 Hadamard transform. Equation (14*a*) shows the flow of Hadamard transform computation, followed by multiplication, where *"N"* is an *even integer* and *"g"* takes a value from the set {0, 1}, while $\hat{N}$ is half of *"N"*. Equation (14b), on the other hand, shows the flow if we first multiply by "MF_{00}" and then divide by 2. It is clear that we may get a wrong result when *g* = 1. What we need to do is to compute *"g"* and make the proper correction, by subtracting "hMF_{00}" (hMF_{00} is half MF_{00}), when *g* = 1.

$$MF_{00}\left(w_{00} \pm w_{01} \pm \cdots\right)/2 = MF_{00}\left(N + g\right)/2 = MF_{00} \cdot \hat{N} \quad (14a)$$

$$\left(w_{00} \cdot MF_{00} \pm w_{01} \cdot MF_{00} \pm \cdots\right)/2 = \left(N \cdot MF_{00} + g \cdot MF_{00}\right)/2 = MF_{00} \cdot \hat{N} + g \cdot hMF_{00} \quad (14b)$$

Our second improvement comes from rewriting the quantization equations (10) and (11), without the absolute value and resigning processes, by introducing new values of *'f'* in case of negative elements W_{ij} . Equations (10) and (11) can be rewritten in integer arithmetic as shown below in equations (15) and (16) respectively.

$$Z_{ij} = \left(W_{ij} . MF_{ij} + f\right) >> qbit \quad (15)$$

$$Z_{ij} = \left(Y_{ij} . MF_{00} + 2f\right) >> qbits + 1 \quad (16)$$

For positive values of W_{ij}, *f* equals $\left(2^{qbits}/3\right)$ for Intra mode and $\left(2^{qbits}/6\right)$ for Inter mode, while for negative values of W_{ij}, it equals $2 * \left(2^{qbits}/3\right)$for Intra mode and $5 * \left(2^{qbits}/6\right)$ for Inter mode.

Equations (15) and (16) are modified from the form written in standard context [2], [3], and [6]. The absolute value and resigning operations are removed by using different values of *f* for negative numbers as described above. Even if this adds complexity in the calculation of *f*, it is beneficial since it removes two additional stages (absolute value and resigning operation) from the critical path.

The suggested modifications in equations (15) and (16) are based on the implementation method of *'round'* process in equations (7) and (8). The principal is simple: division by a number of the form 2^{qbits} can be implemented by an arithmetic right shift operation. However, since the right shift operation always results in a value rounded down to the nearest smallest integer, we need to add a certain value (*f* in our case) to the number before the right shift operation, to get results rounded to the nearest largest integer. This principal is valid for division of positive numbers. In case of negative numbers, the absolute value of the number should be obtained before adding the value *'f'* and the result of right shift operation is then resigned to get back the negative sign. What we suggested here is to use different values of *'f'* for negative W_{ij} elements, in a way that removes the need for absolute value and resigning processes. The new values of *'f'* are calculated by equations (17) and (18) for intra and inter modes respectively.

$$f^{+}_{\text{intra}} = 2^{qbits} / 3 \quad (17a)$$

$$f^{-}_{\text{intra}} = 2^{qbits} - \left(2^{qbits} / 3\right) \quad (17b)$$

$$f^{+}_{\text{inter}} = 2^{qbits} / 6 \quad (18a)$$

$$f^{-}_{\text{inter}} = 2^{qbits} - \left(2^{qbits} / 6\right) \quad (18b)$$

The following calculations show how these new values produce results identical to the standard implementation. Let us represent the term $W_{ij}.MF_{ij}$ for negative values of W_{ij} as

$$\left(-n_{ij} * 2^{qbits} - x_{ij} * 2^{qbits}\right)$$

where n_{ij} is a positive integer and x_{ij} is a positive fraction. For the *'f'* value $2^{qbits}/3$, equation (10) can be calculated as shown below

$$Z_{ij} = \left(\text{abs}\left(-n_{ij} * 2^{qbits} - x_{ij} * 2^{qbits}\right) + 2^{qbits}/3\right) \overset{SHR}{\Rightarrow}$$
$$\begin{cases} n_{ij} + 1 \overset{\text{Resign}}{\Rightarrow} -(n_{ij}+1) & x_{ij} \geq 2/3 \\ n_{ij} \overset{\text{Resign}}{\Rightarrow} -n_{ij} & x_{ij} < 2/3 \end{cases} \quad (19)$$

On the other hand, using equation (15) and equation (17b) to calculate the same value, Z_{ij}, we get the following result.

$$Z_{ij} = \left(\left(-n_{ij} * 2^{qbits} - x_{ij} * 2^{qbits}\right) + \left(2^{qbits} - 2^{qbits}/3\right)\right) \Rightarrow$$
$$\left((1-n_{ij}) * 2^{qbits} - (x_{ij} + 1/3) * 2^{qbits}\right) \overset{SHR}{\Rightarrow}$$
$$\begin{cases} -(n_{ij}+1) & x_{ij} > 2/3 \\ -n_{ij} & x_{ij} \leq 2/3 \end{cases} \quad (20)$$

It is clear that equation (20) produces an incorrect value when $x_{ij} = 2/3$. However, in our case, we will not face this problem, since $\left(-n_{ij} * 2^{qbits} - x_{ij} * 2^{qbits}\right)$ doesn't represent an integer number at $x_{ij} = 2/3$, and since we only deal with integer elements $W_{ij}.MF_{ij}$ we can safely ignore the inconsistency among equations (19) and (20). The same concept applies for the *'f'* value $2^{qbits}/6$, in which case equation (18b) provides the rounding value to be used for negative inter-predicted transform coefficients.

4. HARDWARE IMPLEMENTATION

In this section, we introduce the hardware prototype design for the complete set of transforms and quantization used in H.264 baseline profile, its implementation and the simulation results.

The proposed architecture is called Parallel Paths Architecture (Fig. 1) as transforms and quantization are performed in two parallel paths. It performs the transform and quantization steps on a 4×4 block in one clock cycle, while, for a complete macroblock, it needs 25 clock cycles.

The parallel-paths architecture consists of three modules, core transform, Hadamard transform and quantization that form two paths. In the core transform-quantization path the core transform module was implemented with two stages butterfly-adder blocks as described in [2].

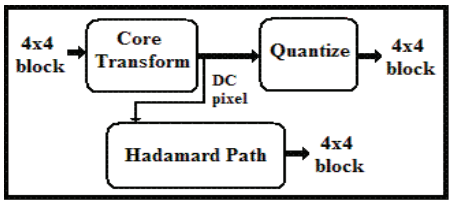


Figure 1. Architecture Design Approach

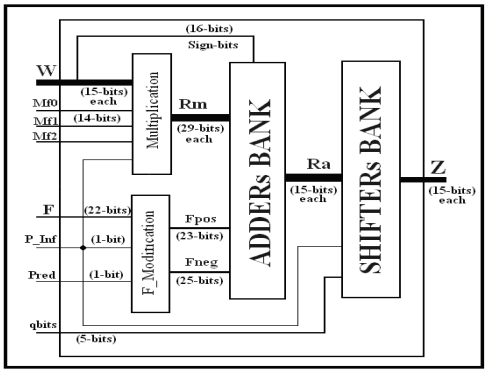
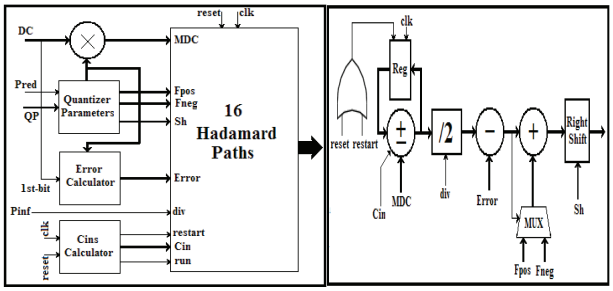


Figure 2. Detailed architecture of the quantization block

The quantization step was implemented according to the modified form as represented by equations (15) and (16). A detailed architecture of the quantization block is presented in Fig. 2. The architecture has hardware redundancy; where the Multiplication, ADDERS BANK, and SHIFTER BANK blocks use 16 multipliers, 16 adders, and 16 shifters respectively. As such, the quantization equations can be calculated for the 16 transform coefficients W_{ij} in parallel. This hardware redundancy increases the area cost, but allows the architecture to run at a higher speed which satisfies the QFHD format target.

The F-Modification sub-block calculates the appropriate value of *f* (see Section 3, eq. 17 & 18) for positive and negative integer elements in parallel, from a base value $2^{qbits}/3$ that was calculated by a separate block. Only shift operations and addition are used to get any one of the four values listed in the previous section. The correct value of *f* to be added is selected through a 2×1 multiplexer controlled by the sign-bit of the integer input value,

therefore a total of 16 multiplexers are used to cover all 4×4 matrix elements.



(a) Architecture block diagram (b) Hadamard path

Figure 3. Block diagram of the Hadamard Unit in Parallel-Paths Architecture

On the other hand, the Hadamard path has been modified so as to include quantization. As main improvement in this approach, we introduce the implementation of the post scale cumulative computation of the Hadamard transform and quantization steps as discussed in Section 3 (equations 12, 13, 14 and 15). Fig. 3.a shows the block diagram of the proposed post-scale cumulative architecture. Only one multiplier is used, since we have one DC coefficient arriving each cycle. The "Error Calculator" unit is responsible for computing the error value (hMF_{00}) mentioned in Section 3 (eq. 14). The critical path, as one can see (Fig.3.b), consists of three 32-bit adders and a multiplier, which is a bit more than the size of adders used in the pre-scale accumulation. However, the use of one multiplier instead of 16 multipliers reduces dramatically the architecture size.

In order to have a clear view of the improvements introduced by our two major proposals, we implemented three architectures for the parallel-paths approach.

In the first implementation (parallel-paths Full Standard, PP_Full_STD), all modules were implemented as specified in the H.264 standard and proposed in [2], [6] and [7]. Next, in the parallel-paths Standard (PP_STD) architecture, the quantization modules were implemented as described in section 3. Finally, the parallel-paths Post-Scale (PP_PS) architecture was implemented by replacing the Hadamard-path by the one described in Fig.3.

All three architectures were prototyped using VHDL language, simulated by Mentor Graphics© ModelSim 6.1® simulation tool and were synthesized using Synopsys Design Compiler®. The target library was the UMC standard cell library, built in 130 nm CMOS technology for worst case conditions. Table 1 summarizes the synthesis results of integrated architecture implementations of the parallel-paths approach.

Table 1. Synthesis Results for Parallel Paths Architectures

	Delay (ns)	Clock Rate (MHz)	Area (#Cells)	Power (mW)
PP_PS	6.388	156.54	65503	69.20
PP_STD	7.127	140.31	89601	87.3
PP_Full_STD	10.090	99.11	86374	136.9

The proposed architecture can easily support QFHD (3840×2160) video with 150Hz frame rate and "4:2:0" color format.

$$clock\_rate = \frac{3840 \times 2160}{4 \times 4} \times 150 \times 1.5 = 116.64\,\text{MHz}$$

Note that the reference implementation (PP_Full_STD) fails to meet this requirement since it can only run at 99.1MHz, while the proposed (PP_PS) provides nearly 34% margin over the necessary rate (116.64MHz).

Comparing the proposed parallel-paths (PP_PS) architecture with the one proposed by Peng et al. [7], one can see that our design performs the transform and quantization steps in one cycle with the same clock rate (156,5 MHz), while our synthesized architecture achieves a power reduction of nearly 20% , as indicated in table 2. An exact comparison is difficult, as the architectures are implemented with different technologies but it is part of our future work to test our design under the 180 nm technology to distinguish between architectural optimizations and technology influences on power reduction.

Table 2. Comparison of the newly proposed architecture with the architecture proposed by Peng et. al. [7]

	Delay (ns)	Clock Rate (MHz)	Area (#Cells)	Power (mW)
PP_PS	6.39	156.50	65503	69.20
Peng [7]	6.39	156.49	30786	83.93

5. CONCLUSION

In this work, we presented a hardware architecture prototype for the transform and quantization stages supported by the H.264 video encoding standard. Experimental results indicate that our design achieve real-time encoding requirements for processing a QFHD video sequence. It provides throughputs as high as 2.4G sample/s Furthermore, a significant gain, of about 20%, is achieved in power consumption. In our future work we plan to implement pipeline versions of the quantization and transform units, which can double the throughput and easily support the UHDTV 7680×4320@60Hz (3G sample/s) format requirement.

6. REFERENCES

[1] ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, 2003.

[2] I. Amer, W. Badawy, and G. Jullien, "Hardware Prototyping for The H.264 4×4 Transformation", *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '04)*, Montreal, Canada, May 2004.

[3] Iain E. G. Richardson, *H.264 and MPEG-4 Video Compression*, John Wiley & Sons Ltd, England, 2003.

[4] T. Stockhammer, M. M. Hannuksela, T. Wiegand, "H.264/AVC in wireless environments", *IEEE Transactions on Circuits und Systems For Video Technology,* Vol. 13, No. 7, July 2003, pp. 657-673.

[5] R. Schafer, T. Wiegand, and H. Schwarz, "The Emerging H.264lAVC Standard", *EBU Technical Review,* January 2003.

[6] I. Amer, W. Badawy, and G. Jullien, "A VLSI Prototype for Hadamard Transform with Application to MPEG-4 Part10", *IEEE International Conference on Multimedia and Expo (ICME '04)*, Taipei, Taiwan, June 2004.

[7] Peng Chungan, Yu Dunshan, Cao Xixin and Sheng Shimin, "A New High Throughput VLSI Architecture for H.264 Transform and Quantization", *IEEE International conference on ASIC* (*ASICON '07)*, October 2007.