15

Variable Bit-Depth Processor for 8×8 Transform and Quantization Coding in H.264/AVC

Gustavo A. Ruiz and Juan A. Michell
Department of Electronics and Computers, University of Cantabria
Spain

1. Introduction

The H.264/AVC (Advanced Video Codec) is the latest standard for video coding established by the Joint Video Team ITU-T VCEG and ISO/IEC MPEG (Wiegand et al., 2003) (Sühring, 2010) (Links, 2010). This standard has many innovations, such as hybrid prediction/transform coding of intra frames and integer transforms (Richardson, 2004). Fig. 1 presents a simplified block diagram of the H.264/AVC encoder with the following main blocks: motion estimation (ME), motion compensation (MC), intra prediction, forward transform (FT), forward quantization (FQ), inverse quantization or re-scaling (IQ), inverse transform (IT), entropy coding and de-blocking filter, among others. Initially, most of the work done on H.264 was oriented toward its software implementation. However, in recent years the contributions to the hardware implementation of H.264 have increased greatly, enabling the implementation of fast architectures for real-time video applications (Lin et al., 2008) (Finchelstein et al., 2009) (Liu et al., 2009).

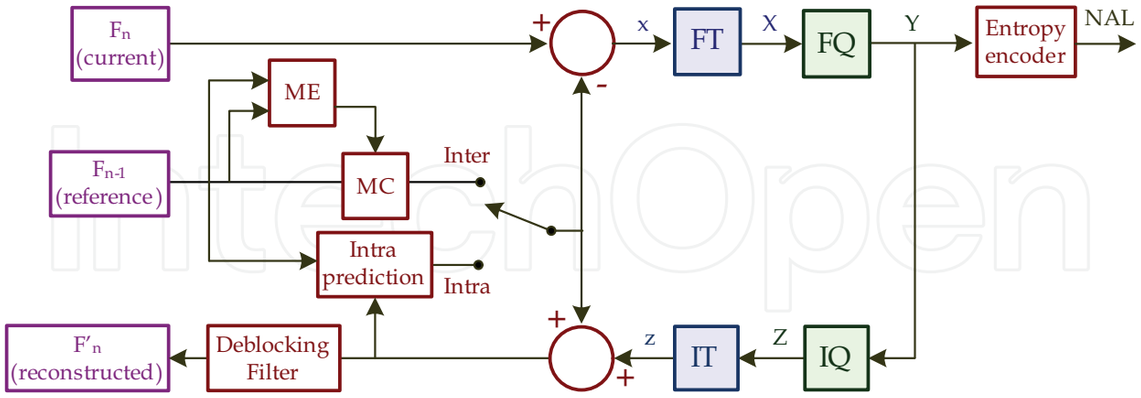


Fig. 1. Diagram of the H.264/AVC encoder.

The initial version of H.264/AVC used a transform hierarchy based on three transforms that are computed in integer arithmetic, two of size 4×4 and one of 2x2. In July 2004, the first amendment to the H.264 standard was presented, named Fidelity Range Extensions (FREext) (JVT, 2004), in which a new set of tools was specified to increase the high-fidelity video encoding efficiency, focusing on professional applications and high-definition videos. One

of the most significant differences between the H.264 FREXt codification and the non-FREXt one is the use of an 8×8 integer transform (Gordon, 2004), which is an integer approximation of the 8×8 2-D Discrete Cosine Transform (DCT), as well as the original 4×4 and 2×2 transforms. The H.264 FREXt enables high quality video by supporting varied chroma sub-sampling formats –4:2:0, 4:2:2 and 4:4:4– with greater color bit-depth ranging from 8-bit up to 14-bit and resolution ranging from QCIF (176x144) to Full HD (1920x1080), both in progressive and interlaced scanning. There are several AVC/H.264 profiles to encode pixels with a bit depth greater than 8 bits: High 10 Profile (8 bits up to 10 bits), high 4:2:2 profile (8 bits up to 10 bits), high 4:4:4 predictive profile (8 bits up to 14 bits), high 10 intra profile (8 bits up to 10 bits), high 4:2:2 intra profile (8 bits up to 10 bits), high 4:4:4 intra profile (8 bits up to 14 bits) and CAVLC 4:4:4 intra profile (8 bits up to 14 bits). Increasing bit depth provides improved accuracy in the compression scheme as well as in motion compensation, in intra prediction and in-loop filtering (Gish, 2002) (Gish, 2003) (Lavier, 2009). Indeed, extensive experimentation proves that the coding efficiency with the largest bit-depth is higher on videos that contain shallow textures and low noise, and perceivable gains exist in the reduction of three kinds of artifacts: contouring, banding and mosquito noise. Currently, bit-depth is especially focused on video quality (Sims et al., 2005). The coding efficiency can be improved by increasing the internal bit depth in relation to the external bit depth used in the video codec (Chujoh & Noda, 2007a, 2007b). Moreover, bit-depth scalability is potentially useful considering that for the foreseeable future, conventional 8-bit and high-bit digital imaging systems will exist simultaneously in the market, providing multiple representations of different bit-depths for the same visual content (Chujoh & Noda, 2006) (Gao & Wu, 2006) (Gao et al., 2010). Other applications of bit-depth are the bit-depth transform of the characteristics for high bit-depth images to maximize the encoding efficiency (Ito et al., 2010), the novel bit-depth expansion method used to remove the contouring effects in smooth regions when mapping low-color bit-depth image to high-color bit-depth (Chen et al., 2009) or the three bit-depth scalable coding architectures compatible with H.264 (Chiang at al., 2009).

This chapter presents a variable bit-depth processor with pipeline architecture for real-time implementation of the complete process for the 8×8 transform and quantization coding in the H.264/AVC. The processor manages different bit-depths – 8 bits up to 14 bits – and quantization parameters (QP) fulfilling the requirements of H.264/AVC. Hardware solutions to reduce its complexity, combined with an efficient implementation, provide a high-speed, high-throughput circuit at a low cost in area. A prototype of the processor, which has been synthesized in a 130nm HCMOS technology, uses 26.5k gates and achieves a maximum speed of 330 MHz with a throughput of 2640 Mpixels/s; this throughput is enough to reach a processing capacity for 1080HD (1920×1088@30fps) real-time video streams.

The remainder of this chapter is organized as follows. Sections 2 and 3 describe the 8×8 transform and quantization in H.264/AVC, providing the necessary mathematical background with special emphasis on describing the effect of the bit-depth in quantization and rescaling expressions. The 8×8 transform provides excellent compression performance in high-resolution video streams with a level of complexity only slightly higher than the 4×4 transform. Its implementation can also be done in terms of additions and shifts and no multiplications are necessary, despite the fact that the coefficients are not powers of 2 in all cases. Quantization and rescaling enable the encoder to control the trade-off between bit-rate and quality. H.264 assumes a bit-depth-dependent scalar quantizer without division

and/or floating arithmetic based on post and pre-scaling matrices. Section 4 describes the proposed architecture for implementing the configurable process of transform and quantization for an 8×8 luma block capable of operating with different bit-depths (8 bits up to 14 bits). This section includes a description of the main modules: 1D configurable forward and inverse transform, 8×8 transpose register and the optimized arithmetic circuit needed to perform the computation of bit-depth-dependent quantization and rescaling in a unified structure. A review of the state-of-the-art of the previous implementations and references is also included. However, most hardware implementations only operate in 8 bits and further bit-depths have not been taken into account. Section 5 shows the characteristics and the performance of the proposed processor as well as comparisons with other published and related implementations. These comparisons are made in terms of area, speed and power.

2. 8×8 Transform in the H.264/AVC

The FREext amendment to H.264 proposes a scheme based on an 8×8 integer approximation of DCT transform to be added to the existing 4×4 transform in order to improve high-definition video compression (Gordon & Wiegand, 2004). This transform provides excellent compression performance in high-resolution video streams with a level of complexity only slightly higher than the 4×4 transform even though the coefficients are not powers of 2 in all the cases. However, it's implemented using additions and shifts and no multiplications are necessary. Moreover it uses integer arithmetic which eliminates the mismatch issues between the encoder and the decoder.

The forward 8×8 integer transform is applied to each block in the residual luminance component (**x**) of the input video stream as follows

$$X = T \cdot x \cdot T^{t} \tag{1}$$

where **T** is a matrix of dimension 8×8 which represents the transform kernel defined as

$$T = \frac{1}{8} \cdot \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix} \tag{2}$$

In the JM reference software (Sühring, 2010), the property of separability of this 8×8 transform is used to implement equation (1) in a separable way as a 1D horizontal (Eq. (3)) transform followed by a 1D vertical (Eq. (4)) transform according to the following equations

$$p = \left(\left(\left(x \cdot T_1^{t}\right) \cdot T_2^{t}\right) \cdot T_3^{t}\right) \tag{3}$$

$$X^{t} = \left(\left(\left(p^{t} \cdot T_1^{t}\right) \cdot T_2^{t}\right) \cdot T_3^{t}\right) \tag{4}$$

Equations (3) and (4) are obtained from the decomposition of **T** as a sparse matrix product of matrices $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$ defined as

$$T_1=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

$$T_2=\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3/2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -3/2 & -1 \\ 0 & 0 & 0 & 0 & 1 & -3/2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 3/2 \end{bmatrix} \tag{6}$$

$$T_3=\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1/4 \\ 0 & 0 & 1 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1/4 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1/4 & 1 & 0 \\ 0 & 0 & 1/2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 & -1 \end{bmatrix} \tag{7}$$

Table 1, which it is directly extracted from the JM reference software, shows the expressions used to compute the 1D transforms involved in equations (3) and (4). In this Table, **IF** denotes the vector of input values (**IF** represents either each row of **x** in equation (3) or each column of **p** in (4)), **OF** denotes the transformed output vector (**OF** represents either each row of **p** in equation (3) or each column of **X** in (4)), and **a** and **b** are internal variables. In a 3-stage butterfly, stage 1 implements the operations involved in $\mathbf{T}_1$, stage 2 implements $\mathbf{T}_2$ and stage 3 implements $\mathbf{T}_3$. The multiplications by the coefficients 1/2, 1/4 and 3/2=1+1/2 are implemented by means of shift-right (>>) operations which cause truncation errors which are propagated through the datapath. To avoid mismatch between the encoder and decoder, the implementation of 1D transform must fulfill the operations specified in the standard. As a result, any implementation of this transform must be in compliance with the arithmetic described in Table 1 and no other alternative is possible.

Stage 1 – $\mathbf{T}_1$	*Stage 2*– $\mathbf{T}_2$	*Stage 3* – $\mathbf{T}_3$
$a_0=IF_0+IF_7$	$b_0=a_0+a_3$	$OF_0=b_0+b_1$
$a_1=IF_1+IF_6$	$b_1=a_1+a_2$	$OF_1=b_4+(b_7>>2)$
$a_2=IF_2+IF_5$	$b_2=a_0-a_3$	$OF_2=b_2+(b_3>>1)$
$a_3=IF_3+IF_4$	$b_3=a_1-a_2$	$OF_3=b_5+(b_6>>2)$
$a_4=IF_0-IF_7$	$b_4=a_5+a_6+((a_4>>1)+a_4)$	$OF_4=b_0-b_1$
$a_5=IF_1-IF_6$	$b_5=a_4-a_7-((a_6>>1)+a_6)$	$OF_5=b_6-(b_5>>2)$
$a_6=IF_2-IF_5$	$b_6=a_4+a_7-((a_5>>1)+a_5)$	$OF_6=(b_2>>1)-b_3$
$a_7=IF_3-IF_4$	$b_7=a_5-a_6+((a_7>>1)+a_7)$	$OF_7=-b_7+(b_4>>2)$

Table 1. Forward 1D transform algorithm extracted from the JM software reference.

The inverse 8×8 integer transform of a block of coefficients of size 8×8 (**Z**) is defined through the equation

$$z=T^{t}\cdot Z\cdot T \tag{8}$$

Likewise to the forward transform, the 8×8 inverse transform can be computed as the concatenation of a 1D horizontal inverse transform (Eq. (9)) and a 1D vertical inverse transform (Eq. (10)) through the decomposition of T as a sparse matrix product of matrices $\mathbf{G}_1$, $\mathbf{G}_2$ and $\mathbf{G}_3$ giving

$$q=\left(\left(\left(Z\cdot G_1\right)\cdot G_2\right)\cdot G_3\right) \tag{9}$$

$$z^{t}=\left(\left(\left(q^{t}\cdot G_1\right)\cdot G_2\right)\cdot G_3\right) \tag{10}$$

The $\mathbf{G}_1$, $\mathbf{G}_2$ and $\mathbf{G}_3$ matrices are defined as

$$G_1=\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 3/2 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & -3/2 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 3/2 & 0 & 1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 3/2 & 0 \\ 0 & -3/2 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{11}$$

$$G_2=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1/4 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/4 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1/4 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

$$G_3=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{13}$$

Table 2 shows the expressions for computing these 1D transforms used in the JM reference software. In a similar way to the forward 1D transform, a 3-stage butterfly structure is used where stage 1 implements the operations specified in $\mathbf{G}_1$, stage 2 in $\mathbf{G}_2$ and stage 3 in $\mathbf{G}_3$. Here, **II** denotes the vector of input values (**II** represents either each file of **Z** in equation (9) or each column of **q** in (10)), **OI** denotes the transformed output vector (**OI** represents either each file of **q** in equation (9) or each column **z** in (10)), and **ia** and **ib** are internal variables.

Stage 1 – $\mathbf{G}_1$	*Stage 2*– $\mathbf{G}_2$	*Stage 3* – $\mathbf{G}_3$
$ia_0=II_0+II_4$	$ib_0=ia_0+ia_6$	$OI_0=ib_0+ib_7$
$ia_1=-II_3+II_5-II_7-(II_7>>1)$	$ib_1=ia_1+(ia_7>>2)$	$OI_1=ib_2+ib_5$
$ia_2=(II_2>>1)-II_6$	$ib_2=ia_4+ia_2$	$OI_2=ib_4+ib_3$
$ia_3=II_1+II_7-II_3-(II_3>>1)$	$ib_3=ia_3+(ia_5>>2)$	$OI_3=ib_6+ib_1$
$ia_4=II_0-II_4$	$ib_4=ia_4-ia_2$	$OI_4=ib_6-ib_1$
$ia_5=-II_1+II_7+II_5+(II_5>>1)$	$ib_5=(ia_3>>2)-ia_5$	$OI_5=ib_4-ib_3$
$ia_6=II_2+(II_6>>1)$	$ib_6=ia_0-ia_6$	$OI_6=ib_2-ib_5$
$ia_7=II_3+II_5+II_1+(II_1>>1)$	$ib_7=-(ia_1>>2)+ia_7$	$OI_7=ib_0-ib_7$

Table 2. Inverse 1D transform algorithm extracted from the JM software reference.

3. Quantization and rescaling in the H.264/AVC

The forward quantization process in H.264/AVC FRExt is performed for the transformed coefficients (**X**) computed in equations (3) and (4) according to the following equations

$$\begin{aligned} \left|Y_{i,j}\right| &= \left(QF_{i,j}\cdot\left|X_{i,j}\right|+\text{lev\_off}\right)>>\text{qbits} \\ \text{sign}\left(Y_{i,j}\right) &= \text{sign}\left(X_{i,j}\right) \end{aligned} \tag{14}$$

where

$$\text{qbits}=QP_{sc}/6+16 \tag{15}$$

In this equation, QP_{sc} is the scaled quantization parameter defined as

$$QP_{sc}=QP+6\cdot(\text{bd}-8) \tag{16}$$

QP takes an integer value (from 0 to 51) and determines the level of coarseness of the quantization process enabling the encoder to control the trade-off between bit rate and

quality. The parameter bd represents the bit-depth video content, $8 \le bd \le 14$. There are lots of professional applications which require higher bit depth support such as studio application and HD application. In H.264/AVC, 7 of 11 profiles support more than 8-bit bit depth starting from High10 which supports 10-bit bit depth. High 444 Predictive and some related profiles support up to 14 bits. As can be seen in equation (16), QP_{sc} depends on the quantization parameter QP as well as bd; note QP_{sc}=QP for bd=8 bits. This means that QP_{sc} can have a value from 0 to 51 when bd=8 and from 36 to 87 for bd=14.
The approximation factor, lev_off, used in equation (14) is defined as

$$\text{lev\_off}=\left(682\cdot \text{intra}+342\cdot \overline{\text{intra}}\right)<<(\text{qbits-11}),\quad \text{intra}\in\{0,1\} \tag{17}$$

where intra=1 is used for intra coefficient quantization and intra=0 for inter coefficient quantization.
The forward quantization matrix, **QF**, is

$$QF=\begin{bmatrix} kf_0 & kf_1 & kf_2 & kf_1 & kf_0 & kf_1 & kf_2 & kf_1 \\ kf_1 & kf_3 & kf_4 & kf_3 & kf_1 & kf_3 & kf_4 & kf_3 \\ kf_2 & kf_4 & kf_5 & kf_4 & kf_2 & kf_4 & kf_5 & kf_4 \\ kf_1 & kf_3 & kf_4 & kf_3 & kf_1 & kf_3 & kf_4 & kf_3 \\ kf_0 & kf_1 & kf_2 & kf_1 & kf_0 & kf_1 & kf_2 & kf_1 \\ kf_1 & kf_3 & kf_4 & kf_3 & kf_1 & kf_3 & kf_4 & kf_3 \\ kf_2 & kf_4 & kf_5 & kf_4 & kf_2 & kf_4 & kf_5 & kf_4 \\ kf_1 & kf_3 & kf_4 & kf_3 & kf_1 & kf_3 & kf_4 & kf_3 \end{bmatrix} \tag{18}$$

whose elements are obtained by evaluating the expression

$$kf_m = MF\left(\text{mod}(QP_{sc},6), m\right)\quad ,\quad \forall m\in\{0,1,2,3,4,5\} \tag{19}$$

In this equation, **MF** is the multiplication factor matrix of dimension 6×6, and the term mod(QP_{sc}, 6) and m denote the row and column indices respectively. **MF** is specified as

$$MF=\begin{bmatrix} 13107 & 12222 & 11428 & 16777 & 15481 & 20972 \\ 11916 & 11058 & 14980 & 10826 & 14290 & 19174 \\ 10082 & 9675 & 12710 & 8943 & 11985 & 15978 \\ 9362 & 8931 & 11984 & 8228 & 11259 & 14913 \\ 8192 & 7740 & 10486 & 7346 & 9777 & 13159 \\ 7282 & 6830 & 9118 & 6428 & 8640 & 11570 \end{bmatrix} \tag{20}$$

The inverse quantization or rescaling "re-scales" the quantized transform coefficients (**Y**) coefficients computed in (14). The rescaling process, which is different to that used in the 4×4 transform (Malvar et al., 2006), is defined by the following equation directly extracted from the JM reference software as

$$Z_{i,j}=\left(\left(\left(QI_{i,j}<<4\right)\cdot Y_{i,j}\right)<<(QP_{sc}/6)+1<<5\right)>>6 \tag{21}$$

where **QI** is the rescaling matrix defined as

$$QI=\begin{bmatrix} ki_0 & ki_1 & ki_2 & ki_1 & ki_0 & ki_1 & ki_2 & ki_1 \\ ki_1 & ki_3 & ki_4 & ki_3 & ki_1 & ki_3 & ki_4 & ki_3 \\ ki_2 & ki_4 & ki_5 & ki_4 & ki_2 & ki_4 & ki_5 & ki_4 \\ ki_1 & ki_3 & ki_4 & ki_3 & ki_1 & ki_3 & ki_4 & ki_3 \\ ki_0 & ki_1 & ki_2 & ki_1 & ki_0 & ki_1 & ki_2 & ki_1 \\ ki_1 & ki_3 & ki_4 & ki_3 & ki_1 & ki_3 & ki_4 & ki_3 \\ ki_2 & ki_4 & ki_5 & ki_4 & ki_2 & ki_4 & ki_5 & ki_4 \\ ki_1 & ki_3 & ki_4 & ki_3 & ki_1 & ki_3 & ki_4 & ki_3 \end{bmatrix} \tag{22}$$

whose elements are obtained by evaluating the expression

$$ki_m = MI\left(\mathrm{mod}(QP_{sc},6), m\right), \quad \forall m \in \{0, 1, 2, 3, 4, 5\} \tag{23}$$

Here, **MI** is the rescaling factor matrix specified as

$$MI=\begin{bmatrix} 20 & 19 & 25 & 18 & 24 & 32 \\ 22 & 21 & 28 & 19 & 26 & 35 \\ 26 & 24 & 33 & 23 & 31 & 42 \\ 28 & 26 & 35 & 25 & 33 & 45 \\ 32 & 30 & 40 & 28 & 38 & 51 \\ 36 & 34 & 46 & 32 & 43 & 58 \end{bmatrix} \tag{24}$$

4. Variable bit-depth processor for the 8×8 transform and quantization

Fig. 2 shows the block diagram of the proposed variable bit-depth processor for real-time implementation of the complete process for the 8×8 transform and quantization coding in the H.264/AVC. This processor includes the following main modules: configurable forward and inverse 1D integer transform, bit-depth dependent quantization and rescaling module, and transpose register memory. This architecture, which fulfils the requirements of H.264/AVC FREext, has been conceived to operate with different bit-depth (bd) – 8 bits up to 14 bits with the aim of achieving a high performance with a reduced hardware complexity implementation. In order to provide an efficient processor, hardware solutions have been developed for the different circuit modules. The 8×8 forward and inverse transforms are calculated using the separability property simplifying its architecture to a single configurable 1D forward (FT)/inverse (IT) transform processor and a transpose register array. Forward quantization (FQ) and rescaling (IQ) operations are computed in the same circuit for the different bit-depth requirements. Here, new expressions are proposed allowing efficient hardware implementation by avoiding the sign conversion and minimizing the arithmetic operations involved. Furthermore, an exhaustive analysis in the dynamic range of the datapath was performed to fix the optimum bus widths with the aim of reducing the size of the circuit while avoiding overflow. Finally, the critical paths of the various computing units have been carefully analyzed and balanced using a pipeline scheme in order to maximize the operation frequency without introducing an excessive latency.

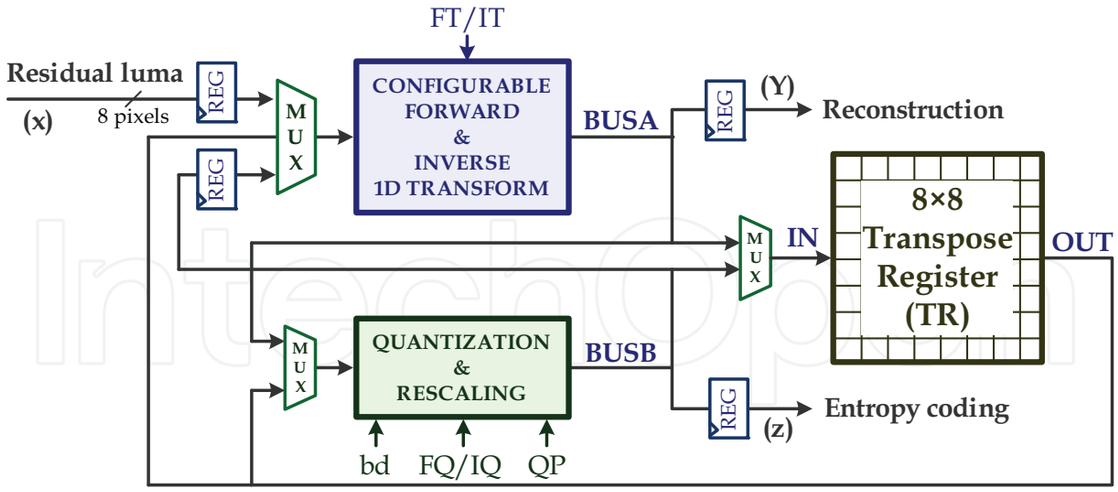


Fig. 2. Block diagram of the variable bit-depth processor.

This circuit processes 8 input data in parallel, starting by reading the residual luminance component (**x**) row by row until the entire 8×8 input block is read. The forward 1D transform module generates the intermediate coefficients **p** to be stored in the transpose register row-wise. After 8 clock cycles, these coefficients are read column-wise and processed again in the 1D transform module. Then, the resulting **X** coefficients are quantized column by column in parallel in the quantization and rescaling module and stored in the transpose register column-wise. On finishing this operation, the quantized coefficients (**Y**) are rescaled row by row and the results (**Z**) are sent to inverse 1D transform whose output data (**q**) are stored in the transpose register row-wise. Finally, the coefficients **q** are fetched to the transpose register column-wise to be processed in the inverse 1D transform to obtain the recovered residual luminance (**z**).

4.1 Forward and Inverse 8×8 transform

The 8×8 transform proposed in FREext for addition to the JVT specification in the H.264/AVC is based on the fact that at SD resolutions and above, the use of block sizes smaller than 8×8 is limited. One of the first papers (Amer et al., 2005) related to this matter was the FPGA pipelined implementation of a simplified 8×8 transform and quantization. Another FPGA implementation of an algebraic integer quantization approach to computing the 8×8 TRANSFROM was presented in (Wahid et al., 2006). (Silva et al., 2007) proposed high-throughput architecture of the forward 8×8 transform to encode high-definition videos in real time with a latency of 5 clock cycles to process 1D transform. This architecture was synthesized in FPGA with a minimum period of 8.13ns and in a TSMC 0.35µm CMOS standard cell technology leading to a period of 8.05ns. Recently, (Park & Ogunfunmi, 2009) presented a reduced and parallel FPGA implementation of an 8×8 integer transform, quantization and scaling for H.264. Here, each pixel is processed one by one on a simplified pipelined architecture without multiplication.

In the adaptive block-size transform of the FREext, different kinds of transforms are required: 8×8 forward/inverse transform, 4×4 forward/inverse transform, 4×4 forward/inverse Hadamard transform and 2×2 forward/inverse Hadamard transform. In order to reduce hardware, diverse configurable data-path architectures to support all of these transforms in

a unified scheme have been proposed. Other examples of this kind of architectures include; the multi-transform processor where the quantization is performed at the pace demanded by the entropy coder in (Bruguera & Osorio, 2006), the low hardware cost suitable for VLSI implementations in (Fan, 2006), the reduced hardware and high latency in (Chao et al., 2007), the high-performance architecture for high-definition applications in (Ma & et. al, 2007), the IP design to be implemented on an ASIP-controlled SoC platform in (Ngo et al., 2008), the high-performance, low-power unified transform architecture in (Choi et al., 2008), the highly parallel joint circuit architecture in (Li et al., 2008), and the fast, high-throughput and cost-effective implementation in (Hwangbo & Kyung, 2010).

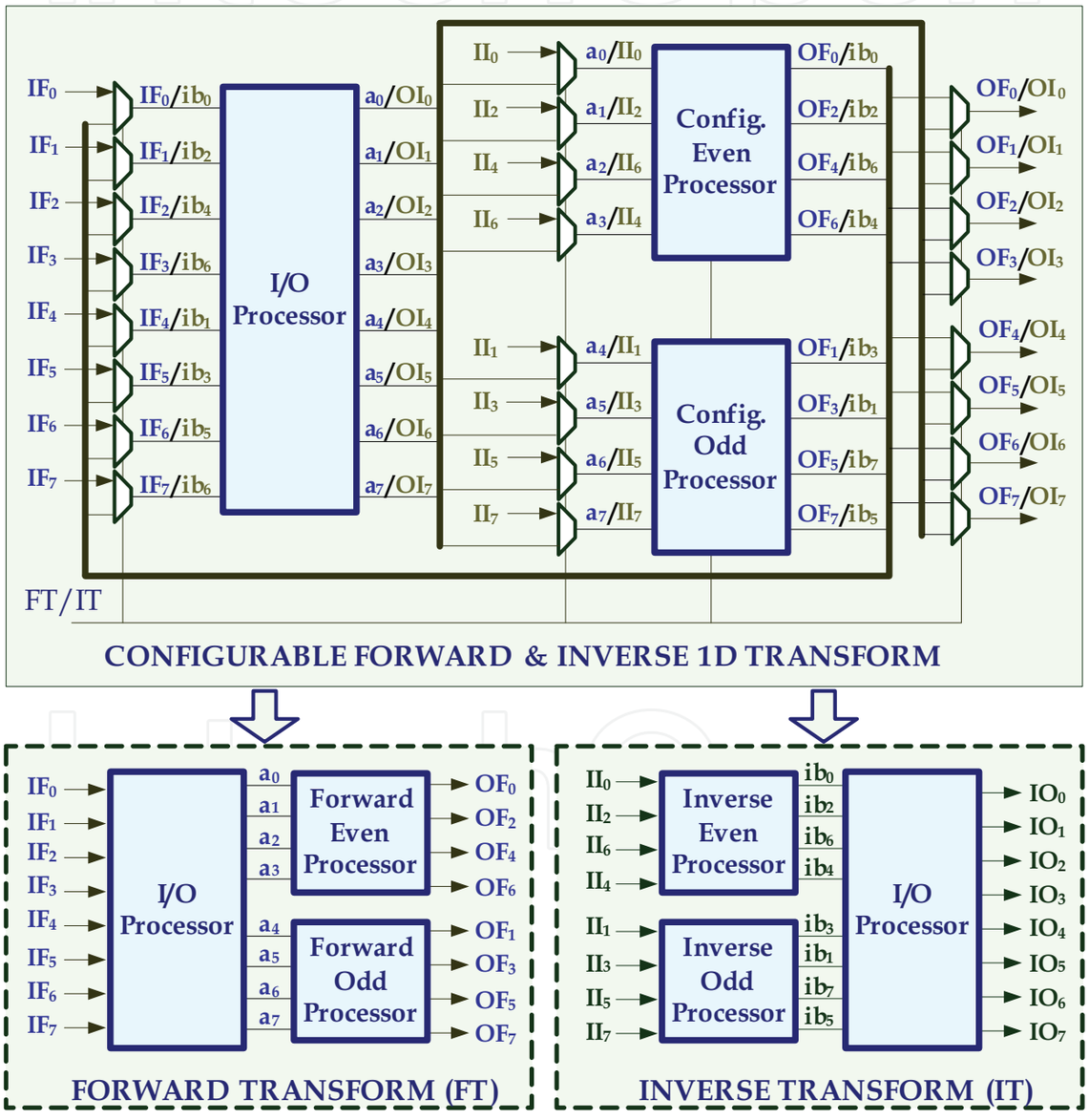


Fig. 3. Block diagram of the forward/inverse transform. The equivalent scheme is also shown for the forward transform (bottom-left) and inverse transform (bottom-right).

Initially, the specifications of H.264 adopted an integer approximation of 4×4, but when transforms are larger, significant compression performance gains have been reported for High-Definition (HD) resolutions. Thus, a new integer transform of 8×8 was proposed in the Fidelity Range Extensions (FRExt) to be added to the previously existing specifications, which were verified in SD resolutions. In fact, the use of block sizes 8x8 and bigger is dominant. Following this assumption, we proposed architecture for computing the 8×8 forward/inverse transform based on a configurable high-throughput 1D processor which has been conceived to implement the arithmetic operations described in Table 1 and Table 2 aiming to fulfill two objectives. First, to avoid mismatches between the encoder and decoder there is no possible alternative in the implementation of the operations other than those specified in these tables, which are directly extracted from the JM reference software. Second, these equations share compatible arithmetic which leads to hardware reduction if a configurable data-path is used. To comply with these prerequisites, arithmetic operations presented in Tables I and II can be implemented in terms of a three-processor architecture that fulfils the requirements of H.264. These processors, as is shown in Fig. 3, are named I/O, even and odd. The operation mode, forward (FT) and inverse (IT), is arranged by multiplexers which select the inputs and modify the inner arithmetic operations of each processor. The schematic at the bottom left in Fig. 3 represents the equivalent scheme for computing the forward 1D transform. In this configuration, the eight elements of **IF** are input to the I/O processor and their outputs run in parallel into the even and odd processors to generate the output **OF**. In the first 1D transform, the input **IF** takes each row of **x** and generates each row of **p** at the output **OF** according to equation (3), and in the second one, each column of **p** is processed to generate each column of **X** according to equation (4). In contrast, the schematic at the bottom left shows the equivalent scheme for the inverse 1D transform. The input data **II** are connected to the even and odd processors while the output data **OI** are generated in the I/O processor. In this configuration, the first inverse 1D transform processes each row of **Z,** generating each column of **q** at the output **OI** according to equation (9), and the second one **q** is read column by column generating each row of **z** according to equation (10).

Fig. 4 shows the data-path of the processors I/O, even and odd. The I/O processor implements the arithmetic operations involved in $\mathbf{T}_1$ (Stage 1 in Table 1) and in $\mathbf{G}_3$ (Stage 3 in Table 2). It is exclusively made up of adders and subtractors where the inputs are properly arranged depending on the operation mode: forward or inverse. Nonetheless, the operations of $\mathbf{T}_2$, $\mathbf{G}_2$, $\mathbf{T}_3$ and $\mathbf{G}_1$ are split up into two processors (even and odd) aiming for the maximum compatibility. As a result, the arithmetic of the even processor varies depending on the operation mode as

$$\begin{matrix} Forward \\ Even \\ \Pr ocessor \end{matrix}\begin{cases} OF_0 = a_0 + a_3 + a_1 + a_2 \\ OF_2 = a_0 - a_3 + \left((a_1 - a_2) >> 1\right) \\ OF_4 = a_0 + a_3 - (a_1 + a_2) \\ OF_6 = \left((a_0 - a_3) >> 1\right) - (a_1 - a_2) \end{cases} \qquad \begin{matrix} Inverse \\ Even \\ \Pr ocessor \end{matrix}\begin{cases} ib_0 = II_0 + II_4 + II_2 - \left(II_6 >> 1\right) \\ ib_2 = II_0 - II_4 + \left(II_2 >> 1\right) - II_6 \\ ib_4 = II_0 - II_4 - \left(II_2 >> 1\right) + II_6 \\ ib_6 = II_0 + II_4 - II_2 + \left(II_6 >> 1\right) \end{cases} \tag{25}$$

This means that this processor is configurable by means of multiplexers used to modify the data path according to the operation mode. In a similar way, the odd processor implements the following equations

$$\begin{matrix} Forward \\ Odd \\ \text{Pr}ocessor \end{matrix} \begin{cases} b_4 = a_5 + a_6 + \left((a_4 >> 1) + a_4\right); & OF_1 = b_4 + \left(b_7 >> 2\right) \\ b_5 = a_4 - a_7 - \left((a_6 >> 1) + a_6\right); & OF_3 = b_5 + \left(b_6 >> 2\right) \\ b_6 = a_4 + a_7 - \left((a_5 >> 1) + a_5\right); & OF_5 = b_6 - \left(b_5 >> 2\right) \\ b_7 = a_5 - a_6 + \left((a_7 >> 1) + a_7\right); & OF_7 = -b_7 + \left(b_4 >> 2\right) \end{cases} \tag{26}$$

$$\begin{matrix} Inverse \\ Odd \\ \text{Pr}ocessor \end{matrix} \begin{cases} ia_1 = II_5 - II_3 - \left((II_7 >> 1) + II_7\right); & ib_1 = ia_1 + \left(ia_7 >> 2\right) \\ ia_3 = II_1 + II_7 - \left((II_3 >> 1) + II_3\right); & ib_3 = ia_3 + \left(ia_5 >> 2\right) \\ ia_5 = II_7 - II_1 + \left((II_5 >> 1) + II_5\right); & ib_5 = -ia_5 + \left(ia_3 >> 2\right) \\ ia_7 = II_3 + II_5 + \left((II_1 >> 1) + II_1\right); & ib_7 = ia_7 - \left(ia_1 >> 2\right) \end{cases} \tag{27}$$

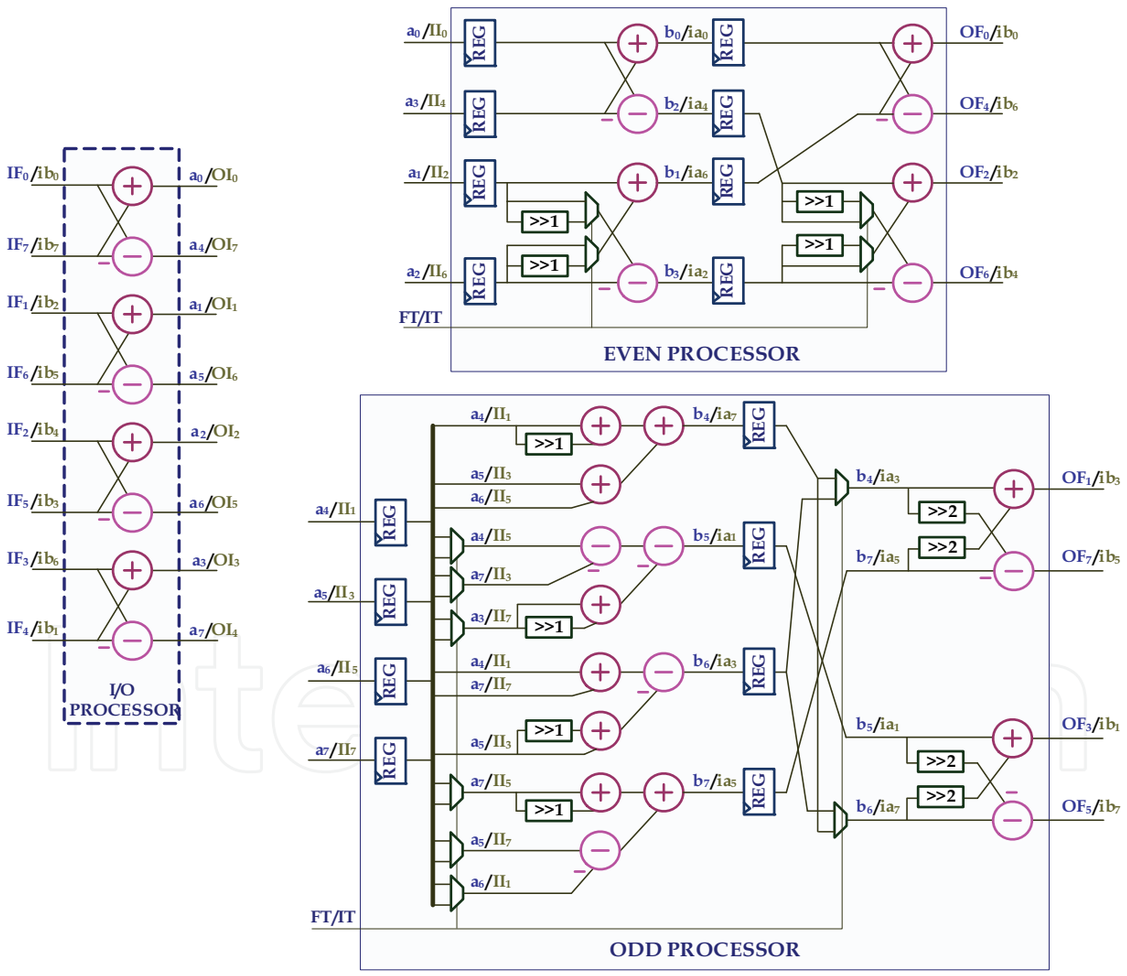


Fig. 4. Schematic of the processors shown in Fig. 3.

The entire circuit to work out the 1D transform takes a total of 32 additions/subtractions and 10 right-shifts that are built by means of data-bus wiring (no additional hardware is necessary). To prevent overflow in the computing of the transform, we consider the biggest

bit-depth of 14 bits for each luminance sample; this means an unsigned integer number from 0 to 16383. However, this processor operates with the residual luminance whose value is ±16383, 15 bits being necessary for its representation. If k represents the input bus width, then k=15 bits for the first forward 1D transform and k=18 for the second one. The intermediate data $a_{0\ to\ 7}$ must be of k+1 bits, $b_{0\ to\ 3}$ of k+2, $b_{4\ to\ 7}$ of k+3, and, finally, the output data of k+3. The range of the coefficients is ±16383·8=±131064 (18 bit) for the first 1D transform, and ±131064·8=±1048512 (21 bit) for the second one. However, the quantization and scaling process increases the data-path by 1 bit, giving input data of 22 bits before calculating the inverse 8×8 transform, this bit width being what limits the data-path of the whole transform module to prevent overflow. This means that all arithmetic in the forward and inverse 1D transform module is performed in 22 bits and the latency is 2 clock cycles.

4.2 Transpose register array

The transpose memory stores 8×8 data and allows simultaneous read and write operations while doing matrix transposition. To achieve this, the 8 input data are read out of the buffer column-wise if the previous intermediate data were written into the buffer row-wise, and vice versa. The transpose buffer based on D-type flip-flops (DFF) (Zhang & Meng, 2009) has been chosen as it is more suitable for pipeline architectures, unlike other proposed architectures based on RAM memories. Indeed, solutions based on a single RAM (Do & Le, 2010) lead to high latency, while those based on duplication of the RAMs (one for processing columns and the other for rows) have a high area cost (Ruiz & Michell, 1998), and those based on bank of SRAMs have a high cost in area (Bojnordi et al., 2006) or in alignment modules (Li et al., 2008).

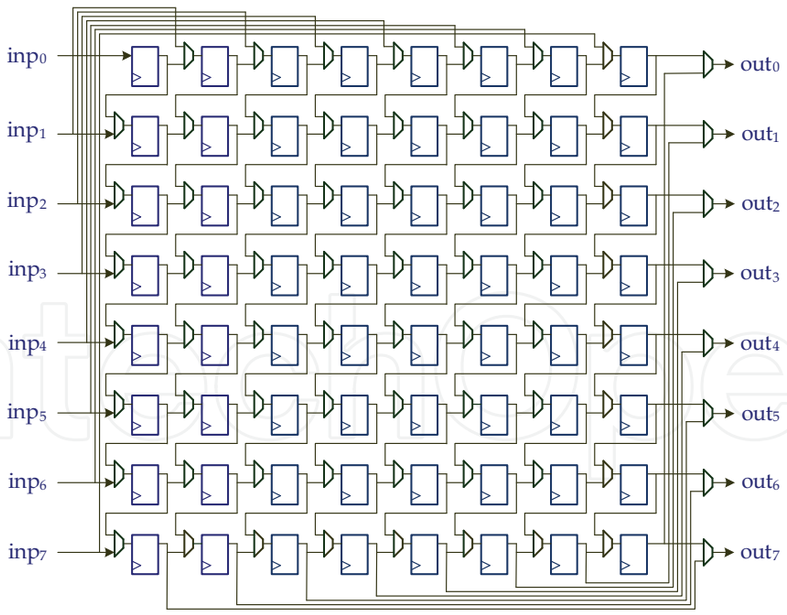


Fig. 5. 8×8 transpose register array.

Fig. 5 shows the schematic of an 8×8 transpose register array of 22 bits each element whose basic cell is a FFD and a multiplexer. Each FFD of the array is interconnected via 2:1 multiplexers forming 8 shift-registers of length 8 either in the horizontal direction (columns) or in the vertical direction (rows). A selection signal controls the direction of shift in the

registers. The loading and shifting mode in the buffer alternates each time a new block of input data is processed: the even (odd) 8×8 block is stored by columns (rows) in the buffer. As a result, the transpose buffer has a parallel input/output structure and the data are transposed on the fly supporting a continuous data flow with the smallest possible size and minimal latency (8 clock cycles).

4.3 Quantization and rescaling

H.264 assumes a scalar quantizer avoiding division and/or floating point arithmetic. Most of the proposed quantization and rescaling hardware solutions attempt to directly implement the expressions defined in the standard, but only a few facilitate its implementation. Moreover, all of them work in 8-bit bit-depth and further bits are not considered. (Amer et al., 2005) presented a simple forward quantizer FPGA design to be run on a Digital Signal Processor. (Wahid et al., 2006) proposed an Algebraic Integer Quantization to reduce the complexity of the quantization and rescaling parameters required for the H.264. The architecture described by (Bruguera and Osorio, 2006) is based on a prediction scheme that allows parallel quantization by detecting zero coefficients to facilitate the entropy encoding. In (Chunganet al., 2007), the multiplier and RAM/ROM were removed by using a 16 parallel shift-adder scheme. An inverse quantizer based on 6-stage pipelined dual issue VLIW-SIMD architecture was proposed in (Lee, J.J. et al., 2008). (Pastuszak, 2008) presented an architecture in a FPGA capable of processing up to 32 coefficients per clock cycle. (Lee & Cho, 2008) proposed a scheme to be applied in several video compression standards such as JPEG, MPEG-1/2/4, H.264 and VC-1 where only one multiplier is used to minimize circuit size. A simplification of the quantization process to reduce overhead logic by removing absolute values leads to a decrease of around 20% in power consumption (Owaida et al., 2009). Another simplification consists of replacing the multiplier with adders and shifters to reduce hardware (Park & Ogunfunmi, 2009). An inverse quantization that adopts three kinds of inverse quantizers based on prediction modes and coefficients used in a H.264/AVC decoder was presented in (Chao et al., 2009). (Husemann et al., 2010) proposed a four forward parallel quantizer architecture implemented in a commercial FPGA board.

We propose a single circuit to compute the forward quantization and rescaling for different bit-depth requirements. In both procedures, multiplication, addition and shifting operations are involved and a configurable architecture enables the same module to perform all the specific operations in order to save hardware. The forward quantization (FQ) operates, cycle by cycle, on the coefficients of each column of the forward 8×8 transform (**X**) and the quantized coefficients (**Y**) are generated according to what is established in equation (14). In this equation, the modulus operation is necessary because the arithmetic operation ">>qbits" performs an integer division with truncation of the result toward zero which causes errors for $X_{i,j}<0$. For example, the integer –3 in a 4-bit two's-complement representation is 1101. The operation –3>>2 should be 0, but 1101>>2 gives –1. To resolve this error, 1<<n–1 must be added to the negative number, where n is the number of right shifts. Thus, (1101+1<<2–1)>>2 is 0. Applying this procedure, the absolute value of $|X_{i,j}|$ can be eliminated from equation (14) by assigning lev_off the same sign as $X_{i,j}$. To do this, a term 1<<qbits–1 must be added when $X_{i,j}<0$. Then, equation (14) can be directly implemented as follows

$$Y_{i,j}=\left(QF_{i,j}\cdot X_{i,j}+\text{lev}\right)>>\text{qbits} \tag{28}$$

where

$$lev = \begin{cases} lev\_off(+) = lev\_off, & for\ X_{i,j} > 0 \\ lev\_off(-) = 1 << qbits - 1 - lev\_off, & for\ X_{i,j} < 0 \end{cases} \tag{29}$$

Therefore, $|X_{i,j}|$ and a subsequent sign conversion should not be necessary in equation (28) which leads to a more efficient hardware implementation than that directly proposed from equation (14). The design to implement equation (28) must be able to manage up to 14-bit depth, that is bd=14. In this case, equation (16) shows that QP_{sc} varies from 36 to 87 as QP does from 0 to 51, and qbits from 22 to 30 according to equation (15). From equations (17) and (29), lev_off(+) for intra mode varies from 1396736 to 357564416, lev_off(–) for intra mode from 2797567 to 716177407, lev_off(+) for inter mode from 700416 to 179306496 and lev_off(–) for inter mode from 3493887 to 894435327. These bounds fix the lev's bit width to 30 bits. Table 3 depicts the definition of lev according to the sign of $X_{i,j}$ and whether intra is 0 or 1, which can be easily implemented by using basic logic and shift operations.

		lev	Binary representation
intra=1	$X_{i,j}\geq 0$	$682<<(5+QP_{SC}/6)$	0...0...0 0 1 0 1 0 1 0 1 0 1 0...0...0
	$X_{i,j}<0$	$-682<<(5+QP_{SC}/6)+(1<<qbits)-1$	0...0...0 1 0 1 0 1 0 1 0 1 0 1...1...1
intra=0	$X_{i,j}\geq 0$	$342<<(5+QP_{SC}/6)$	0...0...0 0 0 1 0 1 0 1 0 1 1 0...0...0
	$X_{i,j}<0$	$-342<<(5+QP_{SC}/6)+(1<<qbits)-1$	0...0...0 1 1 0 1 0 1 0 1 0 0 1...1...1
			Sign extension — 10 — $6+QP_{SC}/6$
			30

Table 3. Definition of lev.

The inverse quantization (IQ) or rescaling specified in (21) can be simplified if this equation is rewritten as follows

$$Z_{i,j} = \left(\left(QI_{i,j} \cdot Y_{i,j} \right) << \left(QP_{sc}/6 \right) + 2 \right) >> 2 \tag{30}$$

Equations (28) and (30) are hardware compatible as they share the same basic arithmetic operations. Fig. 6.a shows the block diagram of the quantizer and rescaling module that is capable of processing 8 coefficients in parallel. It is composed of a control circuit and an 8-way data-path based on a configurable arithmetic unit. The control circuit generates the intermediate parameters needed for the forward quantization or rescaling mode, all of these are obtained from the scaled compression factor (QP_{sc}), the intra value (intra), the operation mode (FQ/IQ) and the operation synchronization (init). These parameters are: lev(+) and lev(–), $\{k_n, k_o, k_p\}$, qbits and qpper defined as

$$qpper = QP_{sc}/6 \tag{31}$$

The three coefficients $\{k_n, k_o, k_p\}$ represent either the quantization multiplication factors $kf_m \in QF_{i,j}$ specified in equations (18), (19) and (20) or the rescaling multiplication factors $ki_m \in QI_{i,j}$ defined in equations (22), (23) and (24). The indexes {n,o,p} take some of these possible values {0, 1, 2}, {1, 3, 4} or {2, 4, 5}. Only three coefficients need to be generated for the 8 arithmetic units because each row or column of the matrix **QF** in (18) or the matrix **QI**

in (22) is composed of three different coefficients. All coefficients are read in a look-up table depending on the operation mode and the value of QP_{sc}.

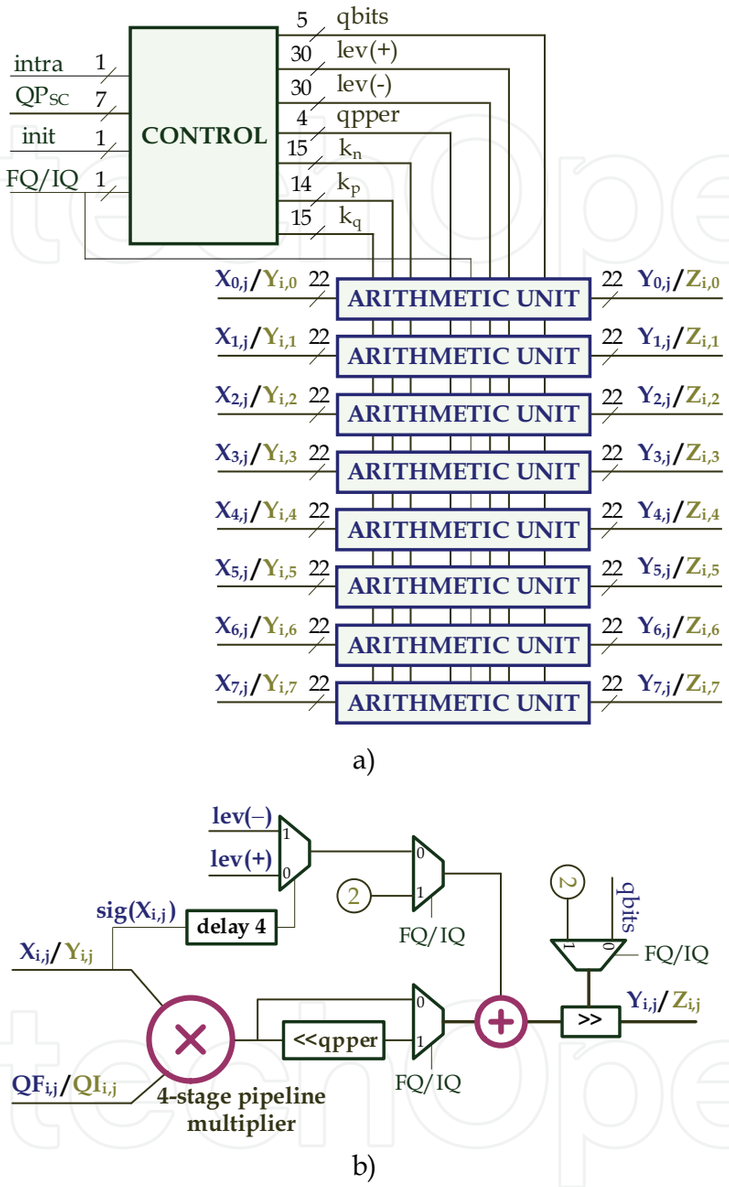


Fig. 6. Configurable forward quantizer and scaling module: a) Block diagram, and b) Schematic of the arithmetic unit.

Fig. 6.b shows a more detailed description of the configurable arithmetic unit. The main arithmetic elements are a multiplier and a adder, and multiplexers and additional logic are used to configure the implementation of equations (28) and (30). The multiplier has a high area cost and delay, so some papers (Michael & Hsu, 2008) (Zhang and et al., 2009) have proposed replacing it with a reduced number of shifts and additions by modifying the **QF** factors to be more suitable for hardware optimization. However, they introduce an error between the quantization and the inverse quantization which leads to a reduction of the

rate-distortion performance. In order to avoid mismatching between encoder and decoder, in our approach an implementation of the whole multiplier is selected, with a pipeline strategy to increase its speed. After an exhaustive analysis, a Wallace-tree 4-stage pipeline multiplier was demonstrated to be the optimal solution to balance the critical path of the multiplier with the critical path of the rest of circuit. In the FQ mode, first the inputs $X_{i,j}$ and $QF_{i,j}$ are multiplied. A multiplexer selects the factor lev(+) or lev(–) to be added to the output of the multiplier depending on the sign of $X_{i,j}$. Here, a delay of 4 clock cycles in the signal of sign($X_{i,j}$) is introduced to compensate for the delay in the multiplier. At the output of the adder, a qbit shift-right (>>) operation is performed to obtain the quantized coefficient $Y_{i,j}$. In the IQ mode, the inputs $Y_{i,j}$ and $QI_{i,j}$ are multiplied. A constant 2 is added to the result and the last >>2 operation generates the scaled coefficients $Z_{i,j}$.

5. ASIC implementation and comparisons

A prototype of the proposed bit-depth processor has been designed and verified using different abstraction levels. Fig. 7 presents the simulation environment used to verify the functional behavior of the proposed architecture by comparing the data processed with those provided by the JM reference software (Sühring, 2010) for different data blocks of input residual luminance. The results of the diverse comparisons performed between the simulation and the reference software indicate that there are no differences between them. Initially, the processor was designed using the CoWare® Signal Processing Worksystem (SPW), editing the block diagram with the elements of the Hardware Design System (HDS) library. The first test bench was made by simulating the design with Simulation Program Builder-Interpreted (SPB-I). The code description in Verilog-RTL was automatically generated by the Verilog RTL Link from the HDS library. A new comparison was performed at this abstraction level to guarantee the correct description of the generated code. Finally, this Verilog description was synthesized using the Synopsys design compiler under HCMOS9 STMicroelectronics 130nm standard cell technology. The resulting circuit contains 26.5k cells with an area of 625700μm² and the estimated maximum operating frequency is 330 MHz. After the logic synthesis, the PrimePowerTM tool was applied to estimate the power consumption, giving 120mW@330MHz (V_{DD}=1.2V). The data throughput is 2640 Mpixels per second. This characteristic enables enough processing capacity for 1080HD (1920x1088@30fps) real-time video streams.

With the proposed architecture, each 8×8 block input data is processed with a latency of 44 clock cycles according to the time scheduling described in Fig. 8. BUSA indicates the output of the transform module, BUSB the output of quantization and scaling module, and IN and OUT are the input and output of the transpose register (TR); all these signals are depicted in Fig. 2. On inputting luma (**x**), it takes 3 clock cycles to generate the coefficients (**p**) and the output coefficients (**X**) are obtained from the 13th clock. These coefficients go to the quantization module and the "quantized" coefficients (**Y**), which are generated from the 18th clock cycle, are stored in the transpose register. In the rescaling process, the data **Y** are read in transpose order to compute the "rescaled" coefficients **Z** from the 31st clock cycle. On processing these coefficients in the 1D transform module, the intermediate data **q** are obtained in the 34st clock cycle. Finally, the recovered residual luminance (**z**) is ready to be processed from the 44th clock cycle and the next luma block can be input in the 49th clock cycle.

For comparison purposes, Table 4 shows the characteristics and the performances of previously published ASIC implementations, although some of them only implement parts

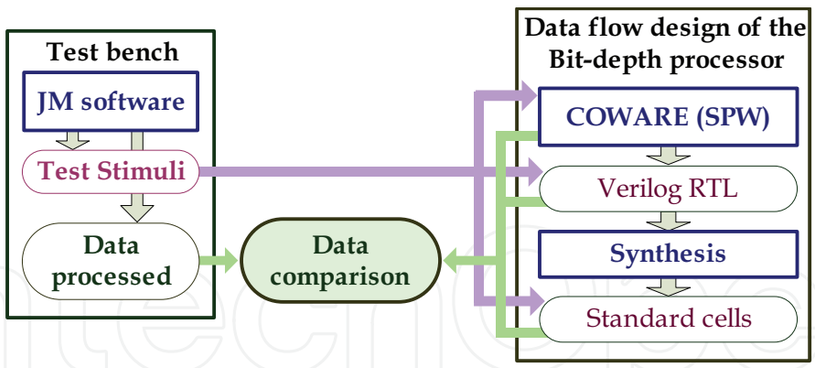


Fig. 7. Block diagram for functional verification of the proposed bit-depth processor.

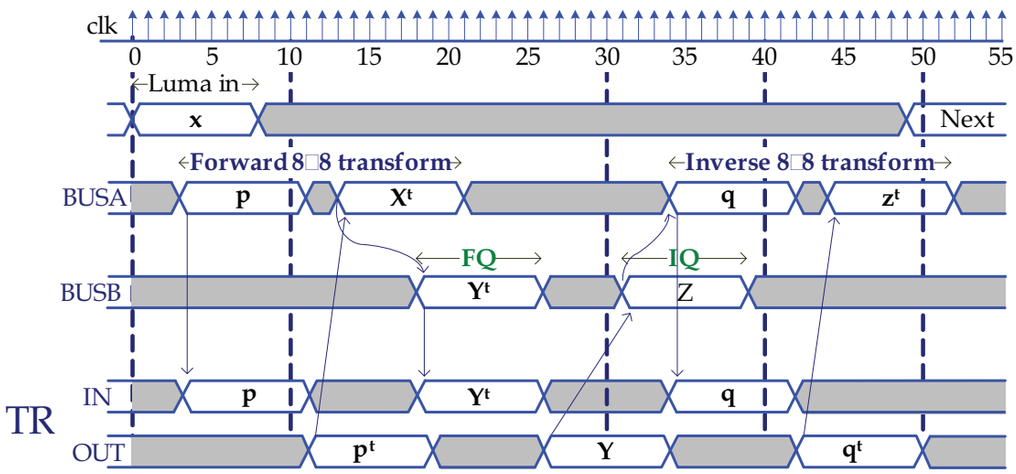


Fig. 8. Time scheduling.

of the H.264/AVC transform coding process. In (Fan, 2006), a cost effective architecture for fast (1-D) 4×4 and 8×8 forward/inverse transform was derived through the Kronecker and direct sum operations. The configurable architecture presented in (Li et al., 2008) supports the six kinds of 4×4 transforms required in the adaptive block-size transform of H.264 in order to more efficiently reuse the data-path; in this architecture, one 8×8 transform can be finished within 16 clock cycles. Based on this reusability property, another unified 4×4 and 8×8 transform architecture is proposed in (Choi at al., 2008). To increase its throughput, 4 units operate in parallel and only 5 clock cycles are needed to perform an 8×8 transform. The low power consumption is because the circuit works at quite low speed (27MHz). A pipeline 8×8 2D forward transform architecture is proposed which is capable of consuming and producing one sample per clock cycle in (Silva et al., 2007). It uses two 1-D transform processors and transpose RAM with a latency of 144 clock cycles. The high-throughput and cost-effective implementation of six different integer transforms is proposed in (Hwangbo & Kyung, 2010). This implementation maximizes the shared hardware and it is able to process 64 input pixels in a two-stage pipelined architecture to compute the direct 8×8 transform or two 4×4 transforms in parallel. Another flexible architecture is presented in (Chao at al., 2007), which is suitable for a H.264 high profile decoder capable of processing a macroblock in 95 clock cycles with the 8×8 inverse transform or only 54 clock cycles without it. The architecture described in (Lee & Cho, 2008) performs the forward 4×4 and 8×8 transform

Ref.	Transform		FQ	bd	Techn.	Area	Speed	Throughput	Power
	Type	Size	IQ		(μm)	(gates)	(MHz)	(Mpixel/s)	
(Fan, 2006)	FWD INV	(1-D) 4, 8	no no	8	TSMC 0.18	6.5k	125	1000	2.5mW @62.5MHz
(Li et al., 2008)	FWD INV	4×4 8×8	no no	8	UMC 0.18	13.6k+ RAM	200	800	N/A
(Choi at al., 2008)	FWD	4×4 8×8	no no	8	AMS 0.35	27k	27	346	9.78mW @27MHz
(Silva et al., 2007)	FWD	8×8	no no	8	TSMC 0.35	33.9k	125	124	N/A
(Chao at al., 2007)	INV	4×4 8×8	no no	8	TSMC 0.18	18.5k	125	860	N/A
(Huang et al., 2008)	FWD INV	4×4 8×8	no no	8	UMC 0.18	39.8k (NAND2)	200	400	38.7mW @50MHz
(Hwangbo & Kyung, 2010)	FWD INV	4×4 8×8	no no	8	UMC 0.18	63.6k	200	3200 6400	86.9mW @200MHz
(Lee & Cho, 2008)	FWD	4×4 8×8	yes no	8	0.18	36.6k+ RAM	103	412	N/A
Pastuszak, 2008)	FWD INV	4×4 8×8	yes yes	8	0.35 0.18	229k 320k	79 76	2528 2432	N/A
(Bruguera et al., 2006)	FWD INV	4×4 8×8	yes yes	8	AMS 0.35	23.8k	67	266	N/A
(Michell et al., 2011)	FWD INV	8×8	yes	8	STM 0.13	29.3k	330	2640	147mW @330MHz
Ours	FWD INV	8×8	yes yes	8 to 14	STM 0.13	26.5k	330	2640	120mW @330MHz

Table 4. Comparison with other architectures for ASIC implementation.

and quantization for unified standard video CODEC (JPEG, MPEG-1/2/4, H.264 and VC-1). A high-throughput architecture which integrates forward transform, quantization, scaling, inverse transform and the sample reconstruction is presented in (Pastuszak, 2008). It uses reconfigurable 4×4 and 8×8 transform architecture and is able to process 32 samples/coefficients per clock cycle. The 8×8 transform is performed in only 2 clock cycles by processing a whole block of 64 input samples through a scheme based on eight 1-D transforms operating in parallel. The quantization and rescaling operate on 32 coefficients in each clock cycle. Although this architecture has low latency, the cost in area is 10 times more than in other proposed designs. In a similar way to (Li et al., 2008), a single data-path for implementing 4×4 and 8×8 forward and inverse transform as well as Hadamard transform is presented in (Bruguera et al., 2006). However, the quantization and rescaling are computed using only one multiplier each and they are performed at the pace demanded by the entropy coder.

In a previous work (Michell et al., 2011), we described a parallel architecture capable of processing 8×8 blocks without interruption with a bit-depth fixed to 8 bit. The latency of 38 clock cycles is achieved by implementing in a pipeline scheme each module used in the transform coding. Indeed, the procesor presented here uses a configurable architecture based on the reusing of different variable bit-depth modules to reduce hardware and power, all of this with a latency of 44 clock cycles. It has been designed attempting to achieve the

maximum throughput at the highest possible speed. To achieve these goals, the pipeline stages have been balanced during the synthesis to maintain the critical path equivalent to 2 adders as a limit, independently of the technology used. Other challenges were the hardware-efficient modifications in the quantization and rescaling module to reduce the arithmetic complexity combined with balanced pipelined multipliers, as it is the more complex arithmetic component, to attain the high performance parameters. According to the results shown in Table 4, our design is the fastest. Its high throughput it is only surpassed by that in (Hwangbo & Kyung, 2010), which processes 16 and 32 input samples in comparison with 8 in our design, but that scheme has a large area cost despite the fact that it only implements the direct transform without quantization and rescaling. The design proposed in (Bruguera et al., 2006) has fewer gates than ours but the quite low speed (67MHz) reduces the throughput to 266Mpixels/s. By observing the differences in the speed and throughput achieved by our processor, we can conclude that these differences cannot only be attributed to the technology used, but are a consequence of the hardware modifications introduced in our design.

6. Conclusions

In July 2004, a new amendment called Fidelity Range Extensions (FRExt) was added to the H.264/AVC as a standardization initiative motivated by the rapidly growing demands focusing on professional applications and high-definition videos. Improvements present in FRExt include a new 8x8 integer transform, the variety of chroma sub-sampling formats and a greater colour bit-depth ranging from 8-bit up to 14-bit. Increasing bit depth provides improved accuracy in the coding efficiency with a reduction of noise and artifacts. Indeed, bit-depth scalability is potentially useful as, in a foreseeable future where different bit-depths will simultaneously coexist in the market, it provides multiple representations of different bit-depths for the same visual content.

This chapter presents a variable bit-depth processor with pipeline architecture for real-time implementation of the complete process for the 8×8 transform and quantization coding in the H.264/AVC. This architecture has been conceived with the aim of achieving a high operation frequency and high throughput without increasing the hardware complexity. Initially, the mathematical expressions of the 8×8 transform and quantization used in the standard H.264/AVC are presented to facilitate the readers' understanding of this matter. A review of the state-of-the-art of the previous implementations and references is also included; here, special emphasis is given to describing the effect of the bit-depth in quantization and rescaling formulas. However, most hardware implementations only operate in 8 bits and further bit-depths have not been taken into account. In order to achieve an efficient implementation of the processor, hardware solutions have been developed for the different circuit modules. A configurable forward and inverse 1D processor and a transpose register array enable an efficient hardware computation of the 8x8 transform. Forward quantization and rescaling operations are computed in the same circuit for different bit-depth requirements and new expressions are included enabling efficient hardware implementation by minimizing the arithmetic operations involved. Finally, the critical paths of the distinct computing units have been carefully analyzed and balanced using a pipeline scheme in order to maximize the operation frequency without introducing an excessive latency. A prototype with the proposed architecture has been synthesized in a 130nm HCMOS technology process which achieves a maximum speed of 330 MHz. The throughput of 2640 Mpixels/s allows real-time video streams of 1080HD (1920×1088@30fps) to be processed.

7. Acknowledgment

We wish to acknowledge the financial help of the Spanish Ministry of Education and Science through TEC2006-12438/TCM received to support this work.

8. References

Amer, W.; Badawy, G. & Jullien, G. (2005). A high-performance hardware implementation of the H.264 simplified 8×8 transformation and quantization. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol.2, pp. II-1137 - II-1140, (March 2005), doi: 10.1109/ICASSP.2005.1415610, ISBN: 0-7803-8874-7

Bojnordi, M.N.; Sedaghati-Mokhtari, N.; Fatemi, O. & Hashemi, M.R. (2006). An efficient self-transposing memory structure for 32-bit video processors. *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 1438–1441, doi: 10.1109/APCCAS.2006.342472, ISBN: 1-4244-0387-1

Bruguera, J.D. & Osorio, R.R. (2006). A unified architecture for H.264 multiple block-size DCT with fast and low cost quantization. *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pp. 407-414, (October 2006), doi: 10.1109/DSD.2006.18, ISBN: 0-7695-2609-8

Chao, T.C.; Tsai, H.H.; Lin, Y.H., Yang, J.F. & Liu, B.D. (2007). A novel design for computing of all transforms in H.264/AVC decoders. *IEEE International Conference on Multimedia and Expo*, pp. 1914-1917, (July 2007), doi: 10.1109/ICME.2007.4285050, ISBN: 1-4244-1016-9

Chao, Y.C.; Wei, S.T.; Liu, B.D. & J.F. Yang, J.F. (2009). Combined CAVLC decoder, inverse quantizer, and transform kernel in compact H.264/AVC decoder. IEEE Transactions on Circuits and Systems for Video Technology, Vol.19, No.1, pp. 53-62, (January 2009), doi: 10.1109/TCSVT.2008.2009251, ISSN: 1051-8215

Cheng, C.H.; Au, O.C.; Liu, C.H. & Yip, K.Y. (2009). *IEEE International Symposium on Circuits and Systems (ISCAS 2009)*, pp. 944-947, doi: 10.1109/ISCAS.2009.5117913, ISBN: 978-1-4244-3827-3

Chiang, J.C. & Kuo, W. T. (2009). Bit-depth scalable video coding using inter-layer prediction from high bit-depth layer. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*, pp. 649-652, doi: 10.1109/ICASSP.2009.4959667, ISBN: 978-1-4244-2353-8

Choi, W.; Park, J. & Lee, S. (2008). A high-performance & low-power unified 4×4 / 8×8 transform architecture for the H.264/AVC Codec. *23rd International Conference Image and Vision Computing*, pp. 1-6, (November 2008), doi: 10.1109/IVCNZ.2008.4762099, ISBN: 9781424437801

Chujoh, T. & Noda, R. (2007a). Internal bit depth increase for coding efficiency. *Joint Video Team*, Doc. VCEG-AE13.doc. Available from http://wftp3.itu.int/av-arch/video-site/0701_Mar/VCEG-AE13.zip

Chujoh, T. & Noda, R. (2007b). Internal bit depth increase except frame memory. *Joint Video Team*, Doc. VCEG-AF07.doc. Available from http://wftp3.itu.int/av-arch/video-site/0704_San/VCEG-AF07.zip

Chungan, P.; Dunshan, Y.; Xixin, C. & Shimin, S. (2007). A new high throughput VLSI architecture for H.264 transform and quantization. *7th International Conference on ASIC (ASICON '07)*, pp.950-953, (October 2007), doi: 10.1109/ICASIC.2007.4415789, ISBN: 978-1-4244-1132-0

Do, T.T.T. & Le, T.M. (2010). High throughput area-efficient SoC-based forward/inverse integer transforms for H.264/AVC. *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4113–4116, doi: 10.1109/ISCAS.2010.5537614, ISBN: 978-1-4244-5308-5

Fan, C.P. (2006). Cost-effective hardware sharing architectures of fast 8×8 and 4×4 integer transforms for H.264/AVC. *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 776–779, (December 2006), doi: 10.1109/APCCAS.2006.342136, ISBN: 1-4244-0387-1

Finchelstein, D.F.; Sze, V. & Chandrakasan, A.P. (2009). Multicore Processing and Efficient On-Chip Caching for H.264 and Future Video Decoders. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.19, No. 11, pp. 1704-1713, doi: 10.1109/TCSVT.2009.2031459, ISSN: 1051-8215

Gao, Y. & Wu, Y. (2006). Applications and requirements for color bit depth scalability. Joint Video Team, Doc. JVT-U049.doc. Available from http://wftp3.itu.int/av-arch/jvt-site/2006_10_Hangzhou/JVT-U049.zip

Gao, Y.; Wu, Y. & Chen, Y. (2009). H.264/Advanced Video Coding (AVC) backward-compatible bit-depth scalable coding. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.19, No.4, (April 2009), pp. 500-510, doi: 10.1109/TCSVT.2009.2014018, ISSN: 1051-8215

Gish, W. (2002). 10-bit and 12-bit sample depth. *Joint Video Team*, Doc. JVT-E048r2.doc. Available from http://wftp3.itu.int/av-arch/jvt-site/2002_10_Geneva/JVT-E048r2.doc

Gish, W. (2003). Extended sample depth: Implementation and characterization. *Joint Video Team*, Doc. JVT-H0.16.doc. Available from http://wftp3.itu.int/av-arch/jvt-site/2003_05_Geneva/JVT-H016.doc

Gordon, S.; Marpe, D. & Wiegand, T. (2004). Simplified use of 8×8 transforms. *Joint Video Team*, Doc. JVT-K028.doc. Available from http://wftp3.itu.int/av-arch/jvt-site/2004_03_Munich/JVT-K028.doc

JVT Joint Video Team of ITU-T and ISO/IEC (2004). Draft text of H.264/AVC fidelity range extensions amendment. *Joint Video Team*, Doc. JVT-L047d9wcm.doc. Available from http://wftp3.itu.int/av-arch/jvt-site/2004_07_Redmond/JVT-L047d9wcm.zip

Huang, C.Y.; Chen, L.F. & Lai, Y.K. (2008). A high-speed 2-D transform architecture with unique kernel for multi-standard video applications. *IEEE International Symposium on Circuits and Systems*, pp. 21-24, (May 2008), doi: 10.1109/ISCAS.2008.4541344, ISBN: 978-2-84813-1

Husemann, R.; Majolo, M.; Guimaraes, V.; Susin, A.; Roesler, V. & Lima, J.V. (2010). Hardware integrated quantization solution for improvement of computational H.264 encoder module. *IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC)*, pp. 316-321, doi: 10.1109/VLSISOC.2010.5642680, ISBN: 978-1-4244-6469-2

Hwangbo, W. & Kyung, C.M. (2010). A multitransform architecture for H.264/AVC high-profile coders. *IEEE Transactions on Multimedia*, Vol.12, No.3, pp. 157-167, (April 2010), doi: 10.1109/TMM.2010.2041099, ISSN: 1520-9210

Ito, T.; Bandoh, Y.; Seishi, T. & Jozawa, H. (2010). A coding method for high bit-depth images based on optimized bit-depth transform. *IEEE International Conference on Image Processing (ICIP)*, pp. 3141-3144, doi: 10.1109/ICIP.2010.5653459, ISBN: 978-1-4244-7994-8

Lee, J.J.; Park, S. & Eum, N.W. (2008). Design of application specific processor for H.264 inverse transform and quantization. *International SoC Design Conference (ISOCC '08)*, pp. II-57 - II-60, (November 2008), doi: 10.1109/SOCDC.2008.4815683, ISBN: 978-1-4244-2598-3

Lavier, P. (2009). Using 10-bit AVC/H.264 encoding with 4:2:2 for broadcast contribution. Ateme company. Confidential report. Available from http://extranet.ateme.com/download.php?file=1114

Lee, S. & Cho, K. (2008). Design of high-performance transform and quantization circuit for unified video CODEC. *IEEE Asia Pacific Conference on Circuits and Systems*, pp. 1450-1453, (November 2008), doi: 10.1109/APCCAS.2008.4746304, ISBN: 0230019544

Lee, Y.; Hong, K. & Kim, S. (2010). An adaptive image bit-depth scaling method for image displays. *IEEE Transactions on Consumer Electronics*, Vol.56, No.1, (March 2010), pp. 141-146, doi: 10.1109/ICCE.2010.5418895, ISSN: 0098-3063

Li, Y.; He, Y. & Mei, S. (2008). A highly parallel joint VLSI architecture for transforms in H.264/AVC. *Journal of Signal Processing Systems*, Vol.50, No.1, (January 2008), pp. 19–32, doi: 10.1007/s11265-007-0111-4, ISSN: 1939-8115

Lin, Y.K.; Li, D.W.; Lin, C.C.; Kuo, T.Y.; Wu, S.J.; Tai, W.C.; Chang, W.C. and Chang, T.S. (2008). A 242mW 10mm2 1080p H.264/AVC High-Profile Encoder Chip. *IEEE International Solid-State Circuits Conference (ISSCC 2008)*, pp. 314-316, doi: 10.1109/ISSCC.2008.4523183, ISBN: 978-1-4244-2010-0

(Links, 2010). Interesting webpage including links to further resources on H.264 and video compression. Available from http://www.vcodex.com/links.html

Liu, Z.; Song, Y.; Shao, M.; Li, S.; Li, L.; Ishiwata, S.; Nakagawa, M.; Goto, S. & Ikenaga, T. (2009). HDTV 1080p H.264/AVC encoder chip design and performance analysis. *IEEE Journal of Solid-State Circuits*, Vol.44, No.2, pp. 594-608, (February 2009), doi: 10.1109/JSSC.2008.2010797, ISSN: 0018-9200

Ma, Y.; Song, Y.; Ikenaga, T. & Goto, S. (2007). A high throughput multiple transform architecture for H.264/AVC fidelity range extensions. *Journal of Semiconductor Technology and Science*, Vol.7, No.4, pp. 247-253, (December 2007), ISSN: 1598-1657

Malvar, H.S.; Hallapuro, A.; Karczewicz, M. & Kerofsky, L. (2003). Low-complexity transform and quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.13, No.7, (July 2003), pp. 598-603, doi: 10.1109/TCSVT.2003.814964, ISSN: 1051-8215

Marpe, D.; Wiegand, T. & Gordon, S. (2005). H.264/MPEG4-AVC fidelity range extensions: Tools, profiles, performance, and application areas. *IEEE Int. Conf. Image Processing*, pp. 593-596, (Sept. 2005), doi: 10.1109/ICIP.2005.1529820, ISBN: 0-7803-9134-9

Michael, M.N. & Hsu, K.W. (2008). A low-power design of quantization for H.264 video coding standard. IEEE International SOC Conference, pp. 201-204, (September 2008), doi: 10.1109/SOCC.2008.4641511, ISBN: 978-1-4244-2596-9

Michell, J.M.; J.M. Solana, J.M. & Ruiz, G.A. (2011). A high-throughput ASIC processor for 8×8 transform coding in H.264/AVC. *Signal Processing: Image Communication*, (in press), doi: 10.1016/j.image.2011.01.001, ISSN: 0923-5965

Ngo, N.T., Do. T.T.T., Le, T.M., Kadam, Y.S. & Bermak, A. (2008). ASIP-controlled inverse integer transform for H.264/AVC compression. *IEEE/IFIP International Symposium on Rapid System Prototyping*, pp. 158-164, (June 2008), doi: 10.1109/RSP.2008.34, ISBN: 978-0-7695-3180-9

Owaida, M.; Koziri, M.; Katsavounidis, I. & Stamoulis, G. (2009) A high performance and low power hardware architecture for the transform & quantization stages in H.264. *IEEE International Conference on Multimedia and Expo (ICME 2009)*, pp. 1102-1105, doi: 10.1109/ICME.2009.5202691, ISBN 978-1-4244-4291-1

Park, J.S. & Ogunfunmi, T. (2009). A new hardware implementation of the H.264 8×8 transform and quantization. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 585-588, doi: 10.1109/ICASSP.2009.4959651, ISBN: 978-1-4244-2354-5, ISSN: 1520-6149

Pastuszak, G. (2008). Transforms and quantization in the high-throughput H.264/AVC encoder based on advanced mode selection. *IEEE Computer Society Annual Symposium on VLSI*, pp. 203-208, (April 2008), doi: 10.1109/ISVLSI.2008.13, ISBN 0-7695-2533-4

Richardson, I.E.G. (2004). H.264 and MPEG-4 Video Compression. John Wiley & Sons (Ed), ISBN: 0-470-84837-5

Ruiz, G.A. & Michell, J.A. (1998). Memory Efficient Programmable Processor Chip for Inverse Haar Transform. *IEEE Transactions on Signal Processing*, Vol.46, No.1, (January 1998), pp 263–268, doi: 10.1109/78.651233, ISSN: 1053-587X

Silva, T.L.; Diniz, C.M.; Vortmann, J.A.; Agostini, L.V.; Susin, A.A. & Bampi, S. (2007). A pipelined 8×8 2-D forward DCT hardware architecture for H.264/AVC high profile encoder. *Proceedings of the 2nd Pacific Conference on Advances in Image and Video Technology*, pp. 5-15, doi: 10.1007/978-3-540-77129-6_5, ISBN: 3-540-77128-X 978-3-540-77128-9

Sims, S.R.F; Mills, J.A. & Topiwala, P.N. (2005). Evaluation of video compression for 8-bit and 12-bit IR data with H.264 fidelity range extensions. *Proc. SPIE the International Society for Optical Engineering*, Vol.5807, pp. 329-340, doi: 10.1117/12.603853, ISBN: 9780819457929

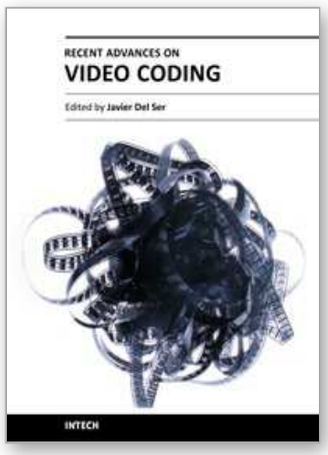
Sühring, K. (2010). H.264/AVC Software Coordination. Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Image Processing Research Department, Berlin, Germany. Available from http://iphome.hhi.de/suehring/tml

Wahid, K.; Dimitrov, V. & Jullien, G. (2006). New Encoding of 8×8 DCT to make H.264 lossless. *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 780-783, doi: 10.1109/APCCAS.2006.342137, ISBN: 0470847549

Wiegand, T.; Sullivan, G.J.; Bjontegaard, G. & Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.13, No.7, (July 2003), pp. 560-576, doi: 10.1109/ICIP.2005.1529820, ISSN: 1051-8215

Zhang, Q. & Meng, N. (2009). A low area pipelined 2-D DCT architecture for JPEG encoder. *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, (August 2009), pp. 747-750, doi: 10.1109/MWSCAS.2009.5235989, ISSN: 1548-3746

Zhang, Y.; Jiang, G. & Yu, M. (2009). Low-complexity quantization for H.264/AVC. *Journal of Real-Time Image Processing*, Vol.4, No.1, pp. 3-12, doi: 10.1007/s11554-008-0098-5, doi: 10.1007/s11554-008-0098-5, ISSN: 1861-8200



Recent Advances on Video Coding

Edited by Dr. Javier Del Ser Lorente

ISBN 978-953-307-181-7
Hard cover, 398 pages
Publisher InTech
Published online 24, June, 2011
Published in print edition June, 2011

This book is intended to attract the attention of practitioners and researchers from industry and academia interested in challenging paradigms of multimedia video coding, with an emphasis on recent technical developments, cross-disciplinary tools and implementations. Given its instructional purpose, the book also overviews recently published video coding standards such as H.264/AVC and SVC from a simulational standpoint. Novel rate control schemes and cross-disciplinary tools for the optimization of diverse aspects related to video coding are also addressed in detail, along with implementation architectures specially tailored for video processing and encoding. The book concludes by exposing new advances in semantic video coding. In summary: this book serves as a technically sounding start point for early-stage researchers and developers willing to join leading-edge research on video coding, processing and multimedia transmission.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen