Low-cost VLSI Architecture Design for Forward Quantization of H.264/AVC

G.A. Ruiz[1] and J.A. Michell
Dpto. de Electrónica y Computadores. Facultad de Ciencias
Universidad de Cantabria. Avda. de Los Castros s/n. 39005 Santander (SPAIN)

ABSTRACT

The H.264/AVC (Advanced Video Codec) is the latest standard for video coding. It assumes a scalar forward quantizer performed at the encoder which can be implemented directly in integer arithmetic. An efficient architecture for the computation of forward quantization of H.264/AVC is presented in this paper. It uses a modification of the quantization operation which reduces the arithmetic operations, and a truncated Booth multiplier based on adaptative statistical approach, which reduces the hardware. The JM reference software's C code has been re-written to analyze the effect of new algorithm and of truncated Booth multiplier. Simulations made up over popular test sequences used in video standardization show the validity of this approach. These results demonstrate that, at low QP, the PSNR is improved between a maximum of +0.81db and a minimum of 0.31db, with a slight increase in the Bit Rate being around 0.8%. Finally, a suitable architecture for VLSI implementation is presented, which reduces in a 26% the area, 32% the power and 21% the critical path delay in comparison with classical implementation. Moreover, it also reduces the area and increase the speed in comparison with architectures presented in references.

Keywords: Quantization, H.264

1. INTRODUCTION

THE H.264/ is the latest standard for video coding established by the Joint Video Team between ITU-T VCEG and ISO/IEC MPEG[1]. H.264 presents a number of features different from the existing standards to support various applications. It assumes a scalar forward quantizer performed at the encoder by a simple scale-and-shift formula which can be implemented directly in integer arithmetic. The step size of the quantizer is controlled with the use of a quantization parameter (QP) which supports 52 different values, from 0 to 51, in increments of one. According to the notation in [2], each transform coefficient with value W_{ij} (i,j= 0 to 3) is quantized in a coefficient Z_{ij} by the following equation:

$$\begin{aligned} |Z_{ij}| &= \left(|W_{ij}| \cdot MF_{ij} + F\right) >> \text{qbits} \\ \text{sign}(Z_{ij}) &= \text{sign}(W_{ij}) \end{aligned} \tag{1}$$

where MF_{ij} is the multiplication factor made up of 6x3 arrays of 14-bit positive integers, qbits=15+floor(QP/6), >> indicates a binary shift right and F is a positive number which can be expressed as F=f << qbits, f being typically in the range 0 to 0.5. Eq. (1) is similar to that used for encoding the 16x16 Intra prediction mode and 4x4 chroma components. In this case, MF_{ij} is replaced by $MF_{0,0}$, F by 2F and qbits by qbits+1. Eq. (1) has been implemented in JM reference software which is available on-line in [3]. In JM reference, f has assigned two values, 1/3 for Intra blocks and 1/6 for Inter blocks.

[1] *ruizrg@unican.es; phone +34 942 20155; fax +34 942 201402*

VLSI Circuits and Systems III, edited by Valentín de Armas Sosa, Kamran Eshraghian, Felix B. Tobajas,
Proc. of SPIE Vol. 6590, 65900Y, (2007) · 0277-786X/07/$18 · doi: 10.1117/12.721493

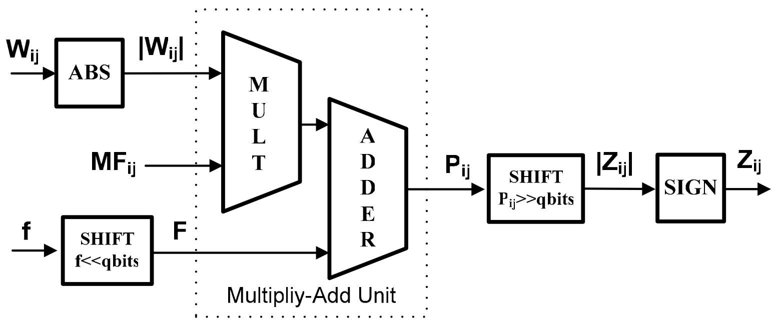


Fig. 1. Quantizer architecture

The forward quantization is not specified in the standard H.264. This allows developers some flexibility in choosing a quantizer design[4]. However, some hardware implementations, as proposed in [5,6], apply directly the quantization expressions of Eq. (1) with any kind of optimization as shown in Fig 1. In this case, the ABS module implements the absolute value of W_{ij}, the multipliy-add unit calculates the term $P_{ij} = (|W_{ij}| \cdot MF_{ij} + F)$ and the final modules make the right-shift and assign to Z_{ij} the same sign of W_{ij}. This paper presents a more efficient quantizer architecture for the computation of forward quantization of H.264. In this architecture, the ABS and SIGN modules are not necessary and an adaptive truncated Booth multiplier is used to reduce hardware.

2. MODIFIED QUANTIZATION OPERATION

In Eq. (1), module $|W_{ij}|$ is necessary because the arithmetic operation ">> qbits" makes an integer division with truncation of the result toward zero which causes errors for $W_{ij} < 0$. 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 must be added to the negative number, n being the number of right shifts. Thus, (1101+(1<<2)) >> 2 is 0. Note that this does not work properly when all the less significant n bits are zero, or in a similar way, when the number is negative and power of two. For example, if n=2 and the number is -4, then (1100 +(1<<2))>>2 is 0 and it should be -1 (1111).

This operation allows $|W_{ij}|$ to be eliminated from Eq. (1) assigning to F the same sign as W_{ij}. To do this, a term 1<<qbits must be added to F when $W_{ij}<0$, resulting in

$$-F+1<<qbits=(1-f)<<qbits \tag{2}$$

Then, Eq (1) can be directly implemented as follows:

$$Z_{ij} = (W_{ij} \cdot MF_{ij} + F^*) >> qbits \tag{3}$$

where

$$F^* = \begin{cases} f << qbits, & \text{for } W_{ij} \geq 0 \\ (1-f) << qbits, & \text{for } W_{ij} < 0 \end{cases} \tag{4}$$

	Sign	f	Definition of F*	18-bit width of F* (qbits=15)
Intra	$W_{ij} \geq 0$	1/3	1/3 << qbits	000010101010101010
	$W_{ij} < 0$	1-1/3=2/3	1/3 << (qbits+1)	000101010101010101
Inter	$Wij \geq 0$	1/6	1/3 << (qbits-1)	000001010101010101
	$Wij < 0$	1-1/6=5/6	1/3 << qbits+1<<(qbits-1)	000110101010101010

Table I. Specification of F*

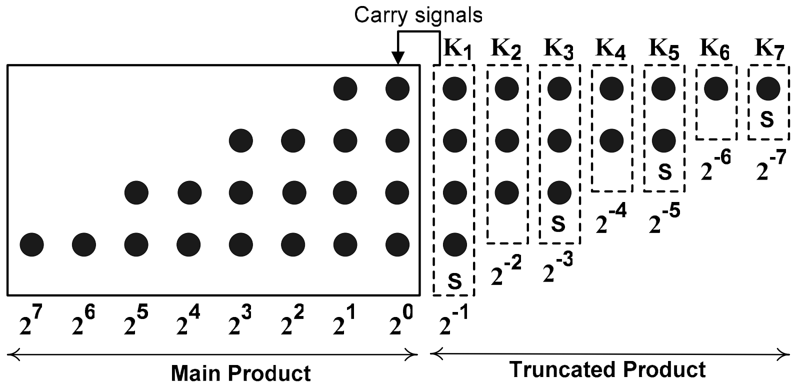


Fig. 2. Truncated 8x8 Booth radix-4 multiplier scheme.

Therefore, $\left|W_{ij}\right|$ and a subsequent sign conversion should not be necessary in Eq. (3) which leads to a more efficient hardware implementation than that proposed in Eq. (1). However, F* must be implemented from Eq. (4). Table I shows the values of f and the definition of F* for different options. F* can be readily generated from number 1/3 and shifted operations. It is noted that f is always positive to be 1/3 or 1/6 and no additions are necessary.

The operation of Eq. (3) provides erroneous results when all less significant qbits in Z_{ij} are zero; it means that Z_{ij} is negative and power of two. The probability of this event is 2^{-qbit} and, in the worst case (qbits=15), its value is $2^{-15} = 1/65536 = 1.52 \cdot 10^{-5}$. Simulations made with real sequences have proven that this error has an insignificant effect in quantization process and, therefore, the proposed method is valid.

2.1 Truncated Booth multiplier

In Eq. (3), the arithmetic operation $W_{ij}MF_{ij}$+F* can be implemented in a single multiplier and the shift operation ">> qbits" is a truncated operation equivalent to eliminating the less significant qbits of the multiplier. Both operations can be efficiently implemented in a truncated multiplier. We focus on the modified Booth's algorithm which is the most popular approach for implementing fast multipliers using parallel encoding.

K_j	**FOREMAN QCIF** 30 frames		**MOBILE CIF** 300 frames		**PARIS CIF** 1065 frames		**TEMPETE CIF** 300 frames		**HIGHWAY CIF** 2000 frames	
j	**PSNR**	**BR**	**PSNR**	**BR**	**PSNR**	**BR**	**PSNR**	**BR**	**PSNR**	**BR**
1	54,11	4882	53,64	20326	53,67	16735	53,96	19480	54.92	16027
2	58,29	4463	57,88	19715	57,83	15244	58,45	18850	60.50	15820
3	65,27	4213	64,66	19090	64,68	14653	65,51	17721	69.60	15448
4	65,30	4107	65,87	18719	65,44	14286	66,65	17590	69.98	15335
5	65,95	4132	65,97	18803	65,61	14336	66,91	17494	70.97	15357
6	65,53	4122	65,95	18779	65,22	14234	66,66	17453	70.56	15307
7	65,36	4107	65,6	18741	65,12	14248	66,48	17427	70.10	15362
8	65,39	4106	65,75	18763	65,11	14239	66,67	17442	70.21	15352
9	65,38	4113	65,69	18751	65,12	14239	66,64	17480	70.30	15367
10	65,38	4114	65,65	18751	65,11	14244	66,60	17461	70.16	15348
11	65,38	4114	65,65	18751	65,11	14244	66,60	17461	70.16	15348
12	65,38	4114	65,65	18751	65,11	14244	66,60	17461	70.16	16027
13	65,38	4114	65,65	18751	65,11	14244	66,60	17461	70.16	15348
14	65,38	4114	65,65	18751	65,11	14244	66,60	17461	70.16	15348
15	65,38	4114	65,65	18751	65,11	14244	66,60	17461	70.16	15348

PSNR means Peak Signal to Noise Ratio (in dB) and BR means Bitrate (in kbit/s)

Table II. Rate distorsion performance for QP=0 using different truncated product.

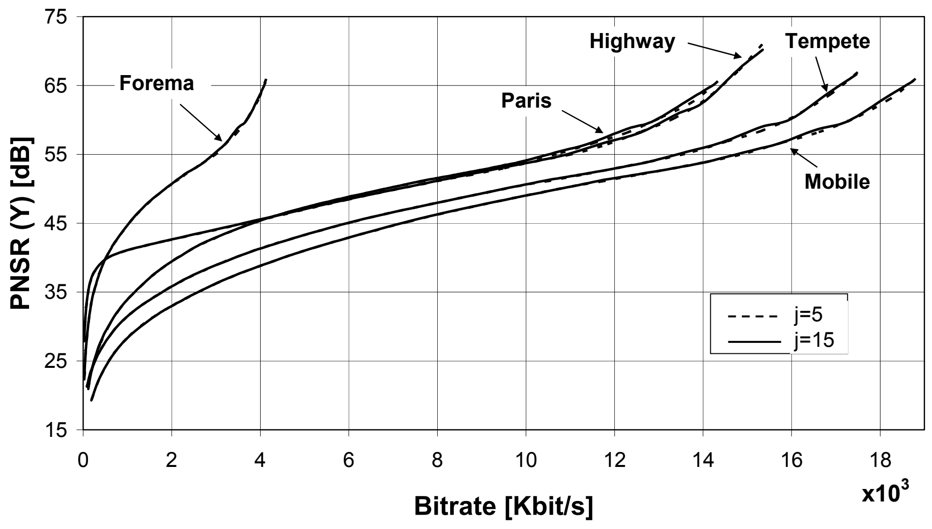


Fig. 3. Rate-distortion curves for j=5 and j=15 (no truncation).

Fig. 2 shows a simple representation of an 8x8 truncated Booth radix-4 multiplier. Each dot is a placeholder for a single bit obtained by a partial product generation circuit and S is the sign conversion bit. All elements can be {0, 1} depending on the result of a partial product selector. In this case, the multiplier's output has been truncated in 7 bits. Thus, partial products are divided into a main product (MP) and a truncated product (TP). The contribution of the TP to the MP is made through the sum of all carry signals generated from the TP which are expressed as:

$$\text{Carry signals} = \sum_{i=1}^{7} K_i 2^{-i} \tag{5}$$

where K_i is the sum of all column dots. This contribution is relatively low in comparison with the MP. Therefore, part of the TP circuitry can be eliminated in order to reduce area and increase the speed of the multiplier. However, an error would be introduced in the resulting product. To reduce this error, several refinements applied to a Booth multiplier have been proposed[7-10]. However, simulations made with these approaches have proven that the adaptive statistical analysis presented in [7] gives the best results. This approach allows the following approximation to be derived:

$$\sum_{i=j}^{7} K_i 2^{-i} \approx 2K_j 2^{-j} = K_j 2^{-j+1} \tag{6}$$

For example, Eq. (5) can be approximated for j=4 as:

$$\text{Carry signals} \approx K_1 2^{-1} + K_2 2^{-2} + (K_3 + K_4) 2^{-3} \tag{7}$$

In this case, the low-error Booth multiplier implementation should only require the columns K_1, K_2, K_3, the dots of K_4 being added to K_3.

2.2 Simulation results

The JM reference software's C code has been re-written to analyze the effect of Eqs. (3) and (4) and of the truncated Booth multiplier for different values of k. Table II shows the simulation results in terms of Peak Signal to Noise Ratio or PSNR (in dB) and bitrate or BR (in kbit/s) for different sequences. The majority of these sequences are popular test sequences used in video standardization. This analysis has been made considering QP=0 which corresponds to maximum bit-rate. Clearly the best PSNR results are obtained for k=5. In this case, the PSNR is improved by a maximum of +0.81dB for Highway sequences to a minimum of +0.31dB for Tempete sequences. No explication is found to justify this improvement in PSNR. The only justification for these results is related to the adaptive error-compensation method used in the multiplier which is based on the statistical approach of partial product bits of adjacent columns. However, this improvement in PSNR is relayed with a slight increase in the lower Bit Rate to 0.8%. The parameter qbits depends linearly on QP/6. For higher QP, the error introduced by the truncated Booth multiplier is drastically reduced as a consequence of the shifting operation in Eq. (1). Fig. 3 shows the rate-distortion curves for different sequences generated for j=5 and for j=15 (no truncated multiplication). Note that only a very slight difference is detected at low QP, the rest of the curve fitting perfectly.

3. QUANTIFIER ARCHITECTURE

Fig. 5 shows an efficient quantizer architecture for H.264 based on a truncated 8-row Booth multiplier. The Booth algorithm is a common approach to the VLSI design of high speed multipliers because the number of additions in multiplication is halved. The modified Booth algorithm proposed by the MacSorley [16] is maybe the most used in hardware implementation because it is fast and requires less area, and their regular structure facilitates efficient implementation in VLSI. In Fig. 5, the truncated Booth multiplier computes on the same carry save structure the expression derived from Eq. (1), $PP_{ij}=(W_{ij}\, MF_{ij}+F^*)<<15$. The 17 different values of MF_{ij} (note that 5243 is duplicated) used in the H.264 reference are listed in Table III. The maximum value of PP_{ij} is computed when $MF_{ij}=13107$ resulting in $PP_{ij} \in [-13107, 13106]$ which it can be described in 15-bit width. The output multiplexer-based shifter allows additional right shift from 0 to a maximum of 8, according to the value of QP/6 and the 16x16 luma/chroma mode. In this scheme, the input data W_{ij} is of 16-bit width, MF_{ij} is of 14-bit ($MF_{ij}>0$) F^* of 14-bit ($F^* > 0$) and output data P_{ij} is of 15-bit arranged for qbits=15. A multiplexer-based shifter is used to generate F^* from the term 1/3<<qbits.

A detailed scheme of truncated Booth multiplier used in quantized is shown in Fig. 6. It is composed by Booth encoders from groups the three bits, partial-product circuit (labeled as SEL), carry save structure based on full adders (FA) and half adders (HA), and a final adder. In Booth encoders, MF_{ij} is partitioned into overlapping groups of three bits and each group is converted into a set of signed digits $\{\pm 2, \pm 1, 0\}$ specified by three signals $\{M, 2M, S\}_k$. These signals select a single partial product D_{km} from W_k, ($W_k=\{W_{ij}\}_k$ in Fig. 6) as

$$D_{km} = \left(W_i \cdot M_k + W_j \cdot 2M_k\right) \oplus S_k \tag{8}$$

Table III shows the Booth decoding for the multiplication factors MF_{ij}. In all of them, the most significative digit is 0 or 1. This imply that the last row of multiplier is simplified in order to reduce hardware necessary to be only one signal, $\{M\}_k$. In scheme of this multiplier, the truncated product generates the intermediate carries to be added to carry save structure and a final carry C_i to be added to final adder. The product sign is defined by the input data W_{ij} to be all MF_{ij} positives.

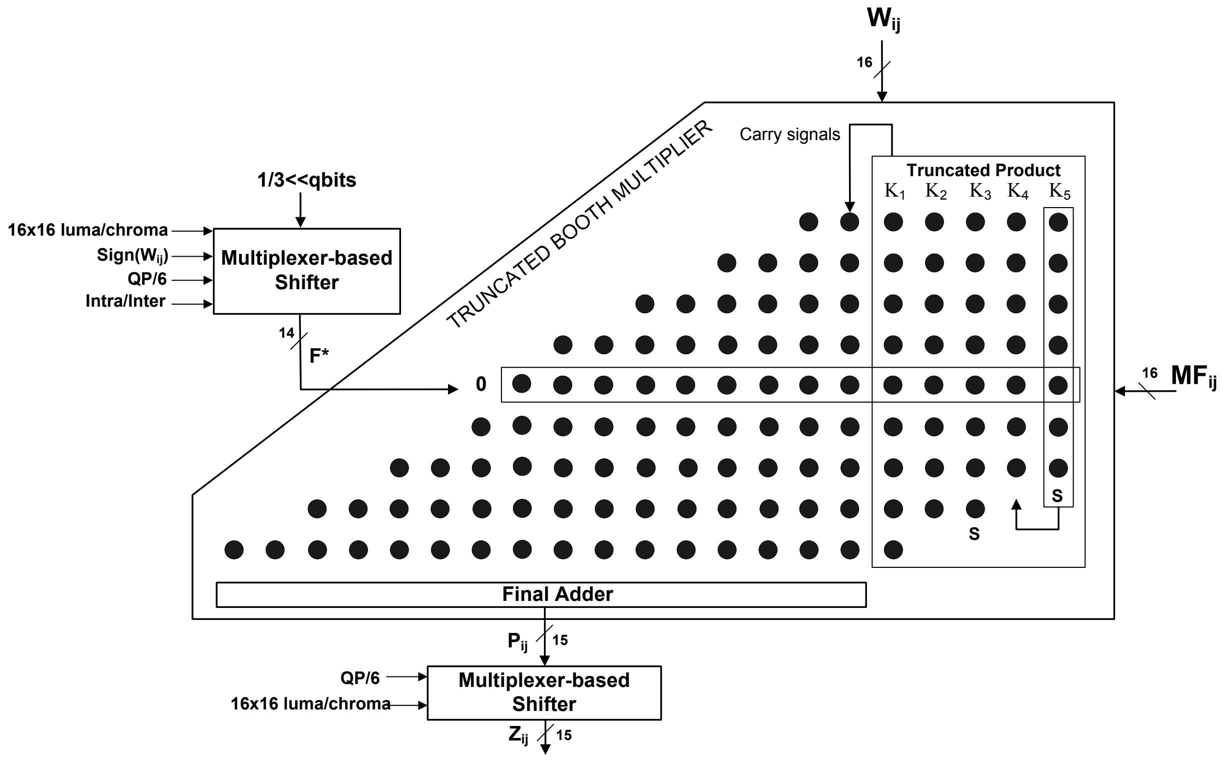


Fig. 5. Scheme of forward quantizer for H.264/AVC

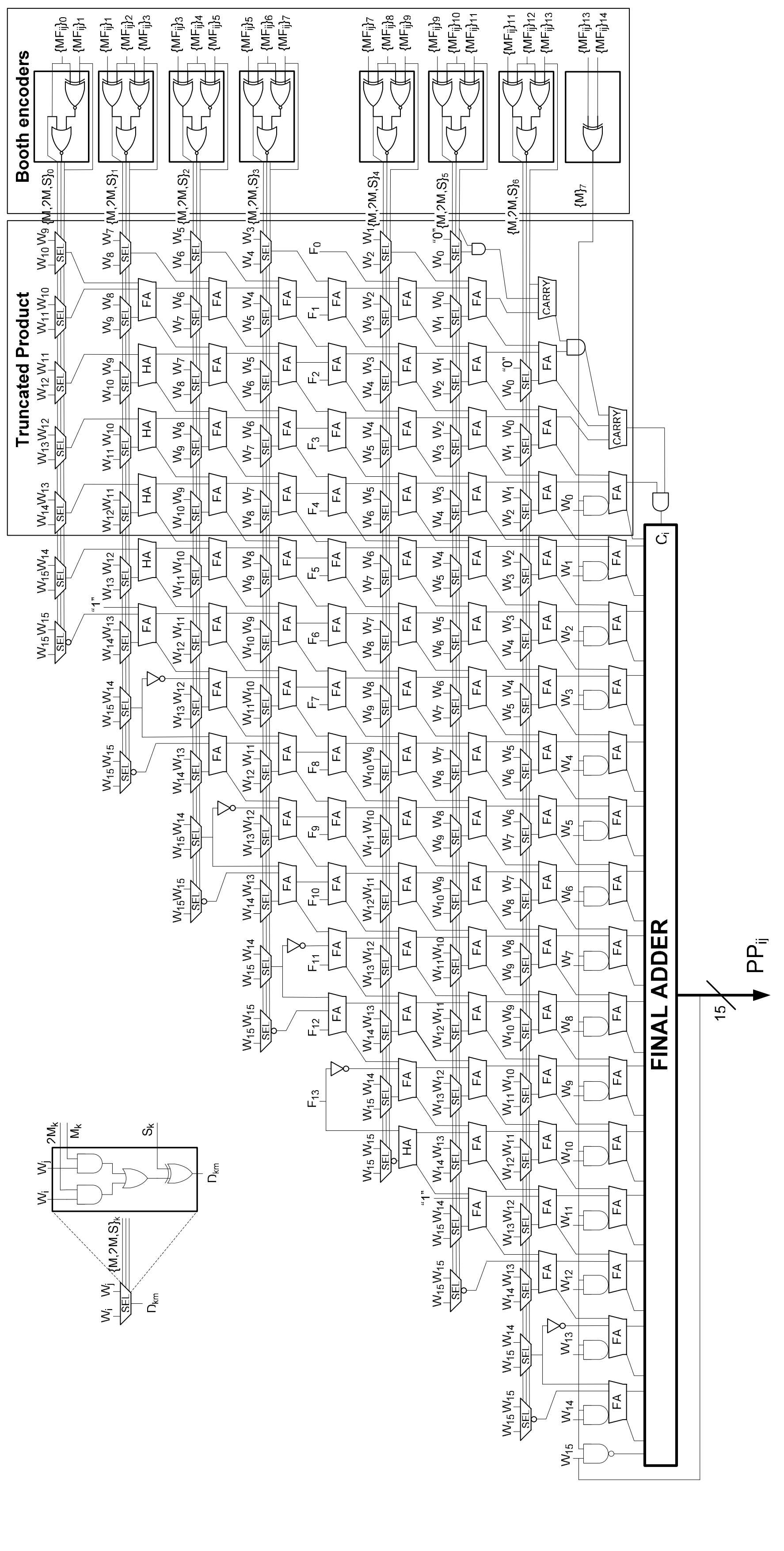


Fig. 6. Truncated 8-row Booth multiplier in carry save arithmetic.

QP/6	(i,j)∈ {(0,0),(2,0),(2,2),(0,2)}		(i,j)∈ {(1,1),(1,3),(3,1),(3,3)}		Other positions	
	Number	**Booth dec.**	**Number**	**Booth dec.**	**Number**	**Booth dec.**
0	13107	$1\bar{1}1\bar{1}1\bar{1}1$	5243	$0110\bar{2}0\bar{1}\bar{1}$	8066	$0200\bar{2}01\bar{2}$
1	11916	$1\bar{1}0\bar{1}\bar{2}1\bar{1}0$	4660	$011\bar{2}1\bar{1}10$	7490	$02\bar{1}1101\bar{2}$
2	10082	$1\bar{2}2\bar{1}2\bar{2}1\bar{2}$	4194	$01002\bar{2}1\bar{2}$	6554	$02\bar{2}2\bar{2}2\bar{1}\bar{2}$
3	9362	$1\bar{2}11\bar{2}11\bar{2}$	3647	$010\bar{2}100\bar{1}$	5825	$012\bar{1}\bar{1}001$
4	8192	$1\bar{2}000000$	3355	$01\bar{1}102\bar{1}\bar{1}$	5243	$011020\bar{1}\bar{1}$
5	7282	$02\bar{1}02\bar{1}02$	2893	$01\bar{1}\bar{1}11\bar{1}1$	4559	$0102\bar{1}10\bar{1}$

Table III. Multiplication factors MF_{ij} and their Booth decoding ($\bar{2}$ means -2 and $\bar{1}$ means -1)

3.1 Sign extension

The addition of partial products in this Booth multiplier must be done with sign bit extension, because it is a signed multiplication. This sign extension is derived from expressions developed in [11,12] which leads to a reduction in area. A formulation which reduces the number of full adders involved in sign extension is presented. For the shake of clarity, Fig. 6 only depicts the extension of sign of each partial product for the particular scheme of multiplier of Fig. 4. Here, $S_{i,16}$ (i=0,1,2,..,6) represents the sign bits for each partial product and $D_{i,j}$ the data bit of partial product. The problem is shown graphically in Fig. 6 where sign must be extended over 7 rows in order to be propagated. The sign SIG of these rows can be written as the result of the following operation:

$$SIG = \sum_{i=16}^{29} S_{0,16} + 2\sum_{i=18}^{29} S_{1,16} + \sum_{i=20}^{29} S_{2,16} + \sum_{i=22}^{29} S_{3,16} + \sum_{i=24}^{29} S_{4,16} + \sum_{i=26}^{29} S_{5,16} + \sum_{i=28}^{29} S_{6,16} \tag{9}$$

Applying the following equivalences:

$$\begin{gathered}\sum_{i=j}^{k} 2^i = 2^{k+1} - 2^j \\ S_{i,16} = 1 - \bar{S}_{i,16}\end{gathered} \tag{10}$$

Then replacing Eq. (10) in (9), SIG becomes

$$\begin{aligned}SIG = {} & (1-\bar{S}_{0,16})(2^{30}-2^{16}) + (1-\bar{S}_{1,16})(2^{30}-2^{18}) + (1-\bar{S}_{2,16})(2^{30}-2^{20}) + \\ & + (1-\bar{S}_{3,16})(2^{30}-2^{22}) + (1-\bar{S}_{4,16})(2^{30}-2^{24}) + (1-\bar{S}_{5,16})(2^{30}-2^{26}) + \\ & + (1-\bar{S}_{6,16})(2^{30}-2^{28})\end{aligned} \tag{11}$$

By reordering this equation, we get

$$\begin{aligned}\mathrm{SIG} = {} & \left[7-\left(\bar{S}_{0,16}+\bar{S}_{1,16}+\bar{S}_{2,16}+\bar{S}_{3,16}+\bar{S}_{4,16}+\bar{S}_{5,16}+\bar{S}_{6,16}\right)\right]2^{30} + \\ & +\bar{S}_{0,16}2^{16}+\bar{S}_{1,16}2^{18}+\bar{S}_{2,16}2^{20}+\bar{S}_{3,16}2^{22}+\bar{S}_{4,16}2^{24}+\bar{S}_{5,16}2^{26}+\bar{S}_{2,16}2^{28} - \\ & -2^{16}-2^{18}-2^{20}-2^{22}-2^{24}-2^{26}-2^{28}\end{aligned} \tag{12}$$

But

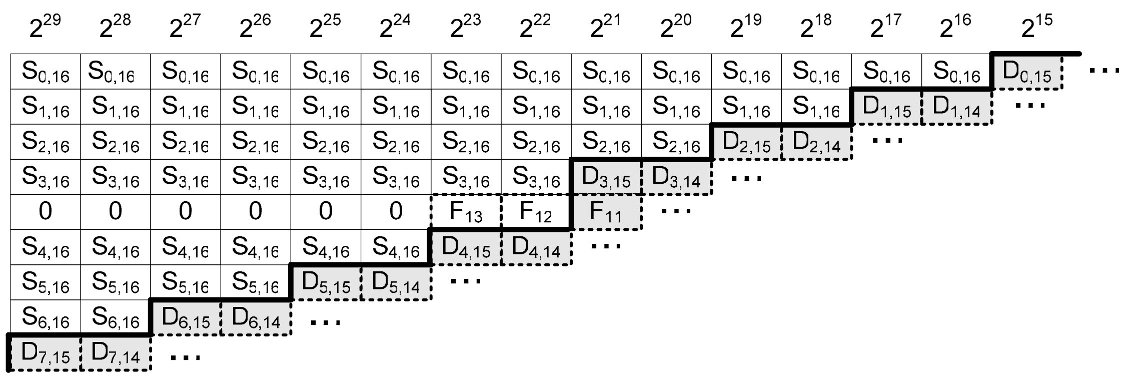
$$2^{30} = \sum_{i=16}^{29} 2^{i} + 2^{16} \tag{13}$$

Then, SIG can therefore be written as:

$$\begin{aligned}\mathrm{SIG} = {} & \left[6-\left(\bar{S}_{0,16}+\bar{S}_{1,16}+\bar{S}_{2,16}+\bar{S}_{3,16}+\bar{S}_{4,16}+\bar{S}_{5,16}+\bar{S}_{6,16}\right)\right]2^{30} + \\ & +\bar{S}_{0,16}2^{16}+\bar{S}_{1,16}2^{18}+\bar{S}_{2,16}2^{20}+\bar{S}_{3,16}2^{22}+\bar{S}_{4,16}2^{24}+\bar{S}_{5,16}2^{26}+\bar{S}_{2,16}2^{28} + \\ & +2^{16}+2^{17}+2^{19}+2^{21}+2^{23}+2^{25}+2^{27}+2^{29}\end{aligned} \tag{14}$$

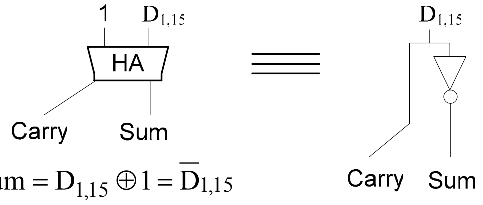
However, every 30th bit can be ignored, resulting in

$$\begin{aligned}\mathrm{SIG} = {} & \bar{S}_{0,16}2^{16}+\bar{S}_{1,16}2^{18}+\bar{S}_{2,16}2^{20}+\bar{S}_{3,16}2^{22}+\bar{S}_{4,16}2^{24}+\bar{S}_{5,16}2^{26}+\bar{S}_{2,16}2^{28} + \\ & +2^{16}+2^{17}+2^{19}+2^{21}+2^{23}+2^{25}+2^{27}+2^{29}\end{aligned} \tag{15}$$



2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	2^{15}	
$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$S_{0,16}$	$D_{0,15}$	...
$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$S_{1,16}$	$D_{1,15}$	$D_{1,14}$	...	
$S_{2,16}$	$S_{2,16}$	$S_{2,16}$	$S_{2,16}$	$S_{2,16}$	$S_{2,16}$	$S_{2,16}$	$S_{2,16}$	$S_{2,16}$	$S_{2,16}$	$D_{2,15}$	$D_{2,14}$	...			
$S_{3,16}$	$S_{3,16}$	$S_{3,16}$	$S_{3,16}$	$S_{3,16}$	$S_{3,16}$	$S_{3,16}$	$S_{3,16}$	$D_{3,15}$	$D_{3,14}$	...					
0	0	0	0	0	0	F_{13}	F_{12}	F_{11}	...						
$S_{4,16}$	$S_{4,16}$	$S_{4,16}$	$S_{4,16}$	$S_{4,16}$	$S_{4,16}$	$D_{4,15}$	$D_{4,14}$	...							
$S_{5,16}$	$S_{5,16}$	$S_{5,16}$	$S_{5,16}$	$D_{5,15}$	$D_{5,14}$	...									
$S_{6,16}$	$S_{6,16}$	$D_{6,15}$	$D_{6,14}$	...											
$D_{7,15}$	$D_{7,14}$	...													

Fig. 6. Section of multiplier with the necessary sign extension of partial products to preserve the correctness of final result.



$$\mathrm{Sum} = D_{1,15} \oplus 1 = \bar{D}_{1,15}$$

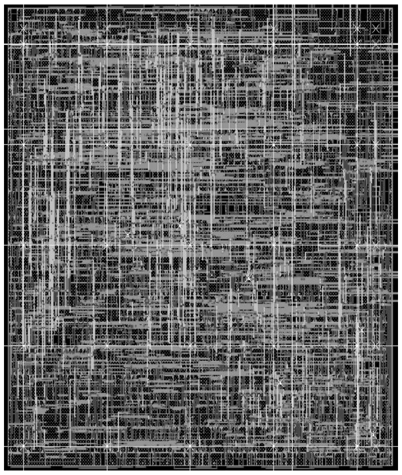
$$\mathrm{Carry} = D_{1,15} \cdot 1 = D_{1,15}$$

Fig. 7. Half adder simplification applied to $D_{1,15}$.

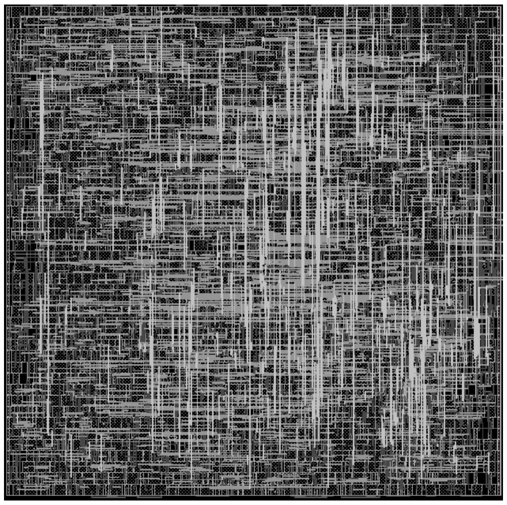
A further reduction is obtained if the independent terms of Eq. (15) are added to carry save structure. For example, the bit 2^{17} can be added to $D_{1,15}$ such as is graphically shown in Fig. 7. As a result, half adders can be replaced by inverters. Furthermore, same simplification is also extended to $\{2^{19}, D_{2,15}\}$, $\{2^{21}, D_{3,15}\}$, $\{2^{23}, F_3\}$, $\{2^{25}, D_{5,15}\}$, $\{2^{27}, D_{6,15}\}$ and $\{2^{29}, D_{7,15}\}$. Sign extension of Eq. (15) and simplification have been both used to reduce hardware in multiplier of Fig. 6.

3.2 Implementation and Comparisons

The architecture presented in Fig. 5 has been described in VERILOG as being easily transferable to a range of silicon fabrication technologies. Moreover, it has been exhaustively verified by comparing the results with test patterns generated using C and MATLAB codes. For the purpose of this research, this architecture has been synthesized by the Synopsys Design Compiler with an AMS 0.35µm standard cell library (3.3 V). The implementation shown in Fig.1 has also been synthesized using the same technology. Layouts of both implementations are shown in Figures 9.a) and 9.b). Synthesis results are shown in Table IV. Clearly, the proposed scheme of Fig. 5 eliminates the need for computation of $\left|W_{ij}\right|$ and a subsequent sign conversion, and the arithmetic operation is performed by a compact truncated Booth multiplier. As a result, it reduces area by 26%, power by 32% and critical path delay by 21%. For comparative analysis,



a)



b)

Figure 9. Layout of a) proposed quantizer (area≈420µm^2*500µm^2) and b) from figure 1 (area=530µm^2*530µm^2).

	Techn.	Pipeline	Multipliers	Number of Cells	Area (µm^2)	Power (mW/MHz)	Critical delay (ns)
Fig. 1	0.35µm	No	1	1974	284500	0.62	16.9
Ours	0.35µm	No	1	1423	210917	0.42	13.35
[5] Area optimized	0.18µm	Yes 4 stages	1	1749			11.7
[5] Speed optimized	0.18µm	No	16	39892			14.7

Table IV. Synthesis results of quantizer comparative analysis.

Table IV also includes the results presented in [5]; only these reference describing quantifier implementations have been found. Here, two quantizer architectures are proposed and their tradeoffs analyzed: 1) optimized for area, which is based on a 4-stage pipelined architecture, and 2) optimized in speed, which is a purely combinational circuit. The latest architecture is conceived to compute for every cycle 16 input data in parallel. Our proposed architecture reduces by 19% the number of cells in comparison with the area optimized scheme and is slightly faster than the speed optimized scheme.

CONCLUSION

A modification of the quantization process and the use of a truncated multiplier have been proposed to implement efficiently the forward quantization of H.264 suitable for VLSI implementation. The proposed architecture presents an important reduction in hardware and power, and an increase in speed, which are achieved by combining a new algorithm for computing Eq. (4) and a compact truncated Booth multiplier. Moreover, some hardware implementations for transform and quantization require several quantizers operating in parallel, 4 in [13], 8 in [14] and 16 in [15]. In these schemes, efficient quantizer architectures are necessary and the proposed quantizer is highly suitable.

ACKNOWLEDMENTS

This research has been supported by funds of the Spanish Ministry of Science and Technology (TIC2006-12438).

REFERENCES

1. T. Wiegand, G.J. Sullivan, G. Bjøntegaard and A. Luthra, A. “Overview of the H.264/AVC video coding standard,” IEEE Trans. on Circuits Syst. and Video Technology, vol. 13, 560-576 (July 2003)
2. I. Richardson, ‘H.264 and MPEG-4 Video Compression’, (Wiley, 2003).
3. [Online] Free on-line software available in http://iphome.hhi.de/suehring/tml/
4. H. Malvar, A. Hallapuro, M. Karczewicz and L. Kerofsky, “Low complexitiy transform and quantization in H.264/AVC,” IEEE Trans. on Circuits Syst. and Video Tech., vol. 13, 598-603, (July 2003)
5. R.C. Kordasiewicz and S. Shirani, “On hardware implementation of DCT and quantization blocks H.264/AVC,” Journal of VLSI Processing, 1-10, (January 2007)
6. O. Tasdizen and I. Hamzaoglu, “A high performance and low cost hardware architecture for H.264 transform and quantization algorithms,” 13th European Signal Processing Conference, Sept. 4-8, (2005)
7. S. J. Jou and H.H. Wang, “Fixed-width multiplier for DSP application,”. Proc. 2000 Int. Conf. Computer Design (ICCD), Austin, TX, 318-322 (Sept. 2000)
8. K.H. Lee and C.S. Rim, “A hardware reduced multiplier for low power design,”, 2nd IEEE Asia Pacific Conference on ASICs, 331-334 (August 2000)
9. K.J. Cho, W.C. Lee, J.G. Chung an K.K. Parhi, “Design of low-error fixe-width modified Booth multiplier,” IEEE Trans. on Very Large Scale Integration (VLSI) Sytems, vol. 12, 522-531 (May 2004)
10. T.B. Juang and S.F. Hsiao, “Low-error carry-free fixed-width multipliers with low-cost compensation circuits,” IEEE Trans. on Circuits and Systems-II, vol. 52, 299-303 (June 2005)
11. O. Salomon, J.M. Green and K. Klar, “General Algorithms for a simplied addition of 2’s complement numbers,” IEEE Journal of Solid_state Circuits, vol. 30 (7), 839-844 (July 1995)
12. Marco Annaratone, ‘Digital CMOS Circuit Design’, Kluwer Academic Publishers (Norwell, MA, USA 1986)
13. 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. on Circuits Syst. and Video Technology, vol. 15 (3), 378-401 (March 2005)

14. Z.Y. Cheng, C.H. Chen, B.D. Liu and J.F. Yang, "High throughput 2-D transform architectures for H.264 advanced video coders," 2004 IEEE Asia-Pacific Conf. on Circuits and Systems, 1141-1144 (Dec. 6-9, 2004)

15. C.P. Fan, "Fast 2-Dimensional 4x4 forward integer transform implementation for H.264/AVC," IEEE Trans. On Circuits and Systems, Vol. 53 (3), 174-177 (, March 2006)

16. O.L. MacSorley, "High-speed arithmetic in binary computers," Proc. Of the IRE, vol. 49, 67-91 (Jan. 1961)