FPGA DESIGN FOR H.264/AVC ENCODER

A. Ben Atitallah(1,2), H. Loukil(2) , N. Masmoudi(2)

(1) University of Sfax, High Institute of Electronics and Communication, BP 868, 3018 Sfax, TUNISIA
(2) LETI laboratory–ENIS, University of Sfax, BP W, 3038 Sfax, TUNISIA
Ahmed.benatitallah@isecs.rnu.tn

Abstract

In this paper, we describe an FPGA H.264/AVC encoder architecture performing at real-time. To reduce the critical path length and to increase throughput, the encoder uses a parallel and pipeline architecture and all modules have been optimized with respect the area cost. Our design is described in VHDL and synthesized to Altera Stratix III FPGA. The throughput of the FPGA architecture reaches a processing rate higher than 177 million of pixels per second at 130 MHz, permitting its use in H.264/AVC standard directed to HDTV.

Index Terms

H.264/AVC, Video coding, VHDL, FPGA architecture.

1. INTRODUCTION

Digital video compression techniques play an important role that enables efficient transmission and storage of multimedia content in bandwidth and storage space limited environment. The H.264/AVC [1, 2, 3] is a video coding standard that has been developed to achieve significant improvements, in the compression performance, over the existing standards. In fact, the high compression performance comes mainly from the prediction techniques that remove spatial and temporal redundancies. To remove spatial redundancy, H.264/AVC intra prediction supports many prediction modes to make better prediction. Inter prediction is enhanced by motion estimation (ME) to remove more temporal redundancy. However, the H.264/AVC coding performance comes at the price of computational complexity. According to the instruction profiling with HDTV specification, H.264/AVC encoding process requires 3600 giga-instructions per second (GIPS) computation and 5570 giga-bytes per second (GBytes/s) memory access. For real-time applications, the acceleration by a dedicated hardware is a must.

This paper focuses on implementing the H.264/AVC encoder in FPGA technology. Indeed, we propose a high throughput rate H.264/AVC encoder in order to support a large band of real time application such as the HDTV (High Definition TV) 720p (1280x720) and 1080i (1920x1088) exploiting advantages of the parallel structures that can be efficiently implemented in hardware using VHDL (VHSIC Hardware Description Language) language. The key point of a parallel architecture is to reduce the number of operations and the ability to achieve fast execution. The rest of the paper is organized as follows: section 2 presents an overview of the H.264/AVC encoder algorithm. The proposed architecture and the FPGA implementation of the H.264/AVC encoder are discussed in section 3. Section 4 presents the synthesis results and the performance evaluations of the H.264/AVC encoder under the Altera FPGA. Finally, a conclusion will be given in section 5.

II. Overview Of H.264/AVC Encoder Algorithm

Figure 1 shows the H.264 encoder scheme that is a hybrid encoder similar to previous standards [1]. A coded video sequence in H.264/AVC consists of a sequence of coded pictures. Each picture is divided into MacroBlocks (MB) of 16x16 pixels. Each MB performs intra and inter prediction mode to find the best predictor in the spatial and temporal domains. There are two kinds of intra prediction modes in H.264. One is intra 4x4 prediction and the other is the intra 16x16 prediction. The inter prediction is implemented by motion estimation prediction on several reference frames. The residual MB is then obtained by subtracting predictor from the original. The residual MB is transformed using an integer transform, and the transform coefficients are quantized followed by zigzag ordering and entropy coding. For coding the residual data block into inter or intra 4x4 prediction mode, the Integer Cosine Transform (ICT), Quantization (Q), Inverse Quantization (IQ) and Inverse ICT (IICT) are applied. But in the intra 16x16 prediction mode, we use both 4x4 ICT and Hadamard transforms with a quantization of the transformed Hadamard coefficients

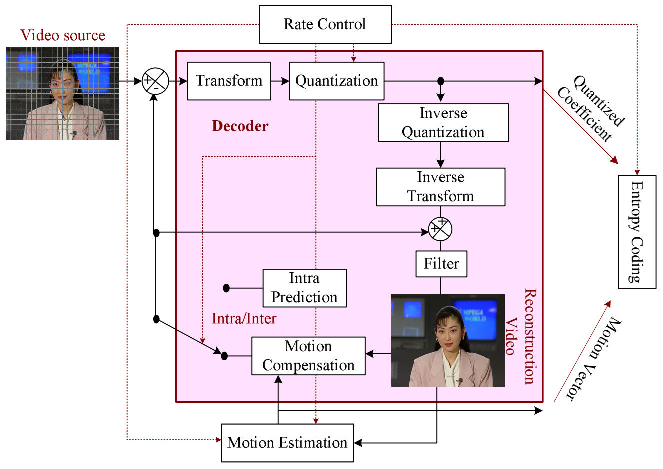


Figure 1. The H.264 encoder scheme

In this section, we present an overview of the different blocks which compose the H.264/AVC encoder such as: prediction mode, ICT/IICT and Q/IQ blocks.

2.1. Prediction modes algorithm

2.1.1. Inter prediction

In fact, motion estimation (ME) is one of the most important operations in H.264/AVC and employs block-based motion estimation to remove temporal redundancy within frames. Thus, it provides coding systems with high compression ratio. The most popular technique for motion estimation is the full search block-matching algorithm (FSBMAs) as sketched in Figure 2.

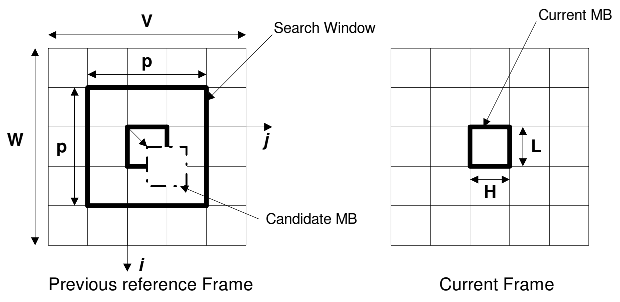
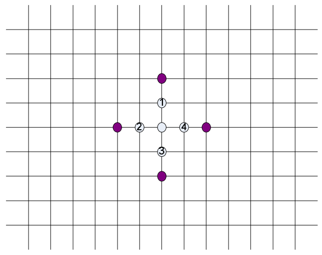


Figure 2. Block-Matching algorithm

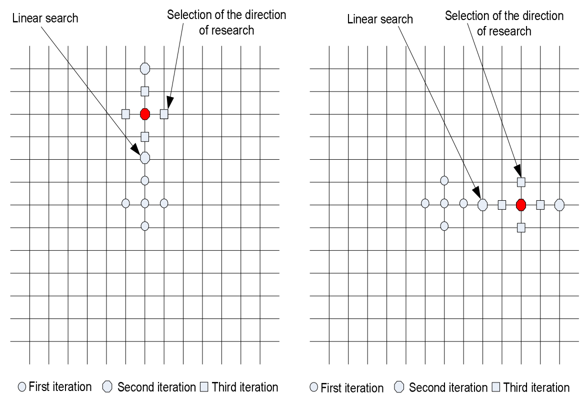
The current frame of a video sequence is divided into 16x16 blocks (current MB). For each of them a block in the previous frame (candidate MB) is exhaustively searched for the best matching within a search window with maximum horizontal and vertical displacements of *p*.
In order to determinate the best MV, The matching algorithm consists in computing an error cost function, usually the Sum of Absolute Differences (SAD) between the MB which is defined by:

$$SAD(m,n)=\sum_{i=0}^{15}\sum_{j=0}^{15}\left|x(i,j)-y(i+m,j+n)\right| \quad (1)$$

Where SAD(m,n) is the distortion of the candidate MB at search position (m,n), x(i,j) means current MB data and y(i,j) stands for search area data. Thus, the FSBMA demand a lot of computation to calculate the distortion for all the $(2p+1)^2$ possible positions of the candidate MBs within the search window whose the pixel size is $(2p+16+1)^2$ pixels. For example, Real Time FSBMA for a 30 Hz CIF (352 x 288 pixels) format with [-16,+15] search window requires 9.3 Giga operations per second (GOPS). This high number of operations makes this approach unsuited for Real Time applications such as 3G mobile phones, CMOS cameras Personal Digital Assistants and wireless surveillance terminals…
Hence, many fast algorithms have been proposed in the literature allowing to reduce of the computational complexity at the price of a slight loss of performance. The basic principle of these algorithms is to divide the search process into a few sequential steps and to choose the next search direction according to the current search result. Based on previous studies [4, 5], the LDPS Line Diamond Parallel Search algorithm (LDPS) provides an acceptable objective quality and speed performance compared to several fast algorithms such as the Three Step Search (TSS) [6], the Diamond Search (DS) [7], the HEXagon-Based Search (HEXBS) [8] and the Nearest Neighbors Search (NNS) [9]. The LDPS [4, 5] search algorithm is illustrated in Figure 3. LDPS exploits the center-biased characteristics of the real world video sequences by using in the initial step, the Small Diamond Search Pattern (SDSP) that is presented in Figure 3.a. The second dynamic pattern improves search on the horizontal and vertical motion components as illustrated in Figure 3.b and c.



(a) Model of the Line Diamond Parallel Search algorithm



(b) Vertical line search (c) Horizontal line search

Figure 3. Operation of Line Diamond Parallel Search

2.1.2. Intra 4x4 prediction mode

There are nine kinds of intra prediction mode for a 4x4 intra block. The intra 4x4 prediction mode is illustrated by Figure 4 where the arrows indicate the direction of prediction in each mode.

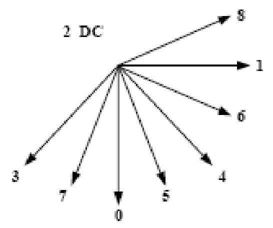


Figure 4. Direction of nine intra 4x4 prediction in H.264/AVC

A 4x4 intra block contains 16 pixels labeled from a to p. The pixels A to M are from the neighboring blocks and are assumed to be already reconstructed. Each intra 4x4 prediction mode generates 16 predicted pixel values (named a to p) using some or all of the neighboring pixels A to M as shown in Figure 5. To encode an intra 4x4 block, we have to opt the best mode with the minimum cost value by computing the sum of absolute transformed difference (SAD) value for all nine candidate modes. After, the residual block, which obtained by the difference between the reference block and best predicted block, is processed by 4x4 integer transform and quantization algorithm and reconstructed by inverse quantization and transform to be the reference of next block.

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

Figure 5. A 4x4 block and neighboring pixels

2.1.3. Intra 16x16 prediction mode

As an alternative to the 4x4 intra prediction mode, the entire 16x16 intra prediction component of a MB may be predicted in one operation. In fact, the current MB is predicted by the 17 pixels from upper MBs and 16 pixels from the left MB. The 16x16 intra prediction has four modes that are calculated for a 16x16 block and is shown in Figure 6.

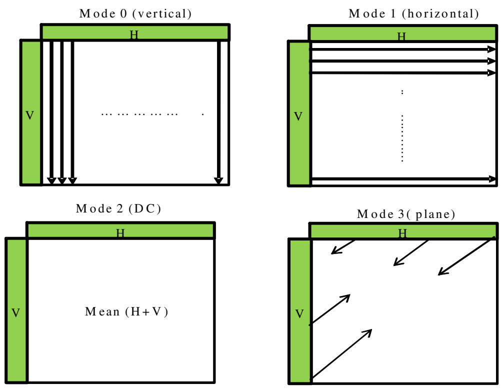


Figure 6. 16x16 intra prediction mode

2.2. 4x4 Integer Transform Algorithm

H.264/AVC adopts transform coding for the prediction error signals. The DCT has been widely used in image and video coding standards, unlike the popular 8x8 DCT utilized in previous standards, while the H.264/AVC encoder is based on a 4x4 ICT which can be computed exactly in integer arithmetic to avoid inverse transform mismatch problems.
There are two types of 4x4 integer transforms for the residual coding. The first one is for luminance residual blocks and is described by (1) [2].

$$Y_{DCT} = MXM^T \tag{2}$$

Where the matrix X is the input 4x4 residual block and M is specified by the following:

$$M = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}$$

With: $a = 1/2, b = \sqrt{1/2}\cos(\pi/8), c = \sqrt{1/2}\cos(3\pi/8)$. Thus, (2) can be factorized in the following form (3) [2]:

$$Y = (CXC^T) \otimes E \tag{3}$$

With:

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & d & -d & -2 \\ 1 & -1 & -1 & 1 \\ d & -2 & 2 & -d \end{bmatrix} \quad and \quad E = \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

Where E is a matrix of scaling factors. The symbol $\otimes$ means that each component of CXC^T is multiplied by the corresponding coefficient in E. To reduce hardware implementation of the transform, the constant d is approximated by 0.5 and the constant b by $\sqrt{2/5}$. The final forward transform becomes (4) [2]:

$$Y = (C_f X C_f^T) \otimes E_f \tag{4}$$

Where:

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \quad and \quad E_f = \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix}$$

So, the scaling matrix E_f is be incorporated into the quantization process. Then $C_f X C_f^T$ becomes the core of a 2-D integer forward transform and contains only 4 coefficients, 1, -1, 2 and -2 that can be implemented by using only additions and shifting operations.
The 4x4 Inverse Integer Cosine Transform (IICT) is very similar to the ICT and the complexity is the same. The coefficient of 1-D inverse transform C_i is given by (5).

$$C_i = \begin{bmatrix} 1 & 1 & 1 & 0.5 \\ 1 & 0.5 & -1 & -1 \\ 1 & -0.5 & -1 & 1 \\ 1 & -1 & 1 & -0.5 \end{bmatrix} \tag{5}$$

The other kind of transform is Hadamard Transform (HT). It is applied to the luminance DC terms in 16x16 intra prediction mode. The Hadamard transform is defined by (6).

$$Y = H_f X H_f^T \tag{6}$$

With:

$$H_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

The Hadamard transform matrix is very similar to the forward transform matrix. The difference is to replace the coefficient 2 by 1 in the transform matrix. The Inverse Hadamard Transform (IHT) is the same as the forward Hadamard transform because the transform matrix is symmetric.

2.3. 4x4 Quantization Algorithm

In H.264/AVC, the quantization matrix E_f which defined by equation (4) is incorporated in the quantization. Then the quantizer mechanisms become complicated because of the requirements for avoiding division and floating point arithmetic. In H.264/AVC, there are two types of quantization algorithm for the 4x4 integer transform. The first one is for the transformed coefficients of luminance residual block. The AC Quantization Operation (ACQ) is shown in (7) [2].

$$Zij = round(Y ij \frac{PF}{QStep}) \tag{7}$$

Where, $Y = (C_f X C_f^T)$ is the unscaled coefficient after integer core transformation, *PF* is the scaling factor of integer transform, *QStep* is the quantization step size and Z_{ij} is the coefficient after quantization. A total of 52 vlues of *QStep* are supported by the standard as presented in Table 1, where *QStep* doubles in size for every increment of 6 in *QP*. Hence, to simplify the arithmetic, the quantization stated in (7) can be rewritten as (8) and implemented *PF/QStep* as a multiplication by a factor *MF* (a multiplication factor) and a right-shift, to avoid the division operations.

$$Z_{ij} = round(Y_{ij} \frac{MF}{2^{qbits}}) \qquad (8)$$

Where

$$\frac{MF}{2^{qbits}} = \frac{PF}{Qstep} \qquad (9)$$

$$qbits = 15 + floor(QP/6) \qquad (10)$$

Table 1. Quantization step sizes in H.264/AVC codec

QP	0	1	2	3	4	5
QStep	0.625	0.6875	0.8125	0.875	1	1.125
QP	6	7	8	9	10	11
QStep	1.25	1.375	1.625	1.75	2	2.25
QP	…	…	…	…	…	…
QStep	…	…	…	…	…	…
QP	48	49	50	51		
QStep	160	…	…	224		

The *MF* value depends on *QP* and the position *(i,j)* of the element in the matrix as shown in Table 2. The factor *MF* remains unchanged for *QP>5* which can be calculated using (11).

$$MF_{QP>5} = MF_{QP=QP\%6} \qquad (11)$$

Then (8) can be represented using integer arithmetic [2] as follows:

$$|Z_{ij}| = (|Y_{ij}|.MF + f) >> qbits \qquad (12)$$

Were *f* is a parameter used to avoid rounding errors and it depends on prediction type of the block and *QP*.

Table 2. The Multiplication factor MF in H.264/AVC

QP	**Positions** (0,0),(2,0),(0,2),(2,2)	**Positions** (1,1),(1,3),(3,1),(3,3)	**Other positions**
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

The Inverse of AC Quantization (IACQ) is defined as:

$$Y_{ij} = Z_{ij}.V_{ij}.2^{floor(QP/6)} \tag{13}$$

Where Z_{ij} is the quantized coefficient, Y_{ij} is a scaled coefficient and V_{ij} is rescaling factor.

The other type of quantization is for DC coefficients of 4x4 Hadamard transform. The DC Quantization (DCQ) is shown in 14.

$$|Z_{ij}| = (|Y_{ij}|.MF_{(0,0)} + 2f) >> qbits + 1 \tag{14}$$
$$sign(Z_{ij}) = sign(Y_{ij})$$

Where $MF_{(0,0)}$ is the multiplication factor for position (0, 0) in Table 2. The inverse of DC quantization (IDCQ) is defined as:

If QP≥12 then:

$$Y_{ij} = Z_{ij}.V_{(0,0)}.2^{floor(QP/6)+2} \tag{15}$$

Otherwise:

$$Y_{ij} = \left[Z_{ij}.V_{(0,0)} + 2^{1-floor(QP/6)}\right] >> (2 - floor(QP/6))$$

3. PROPOSED HARDWARE ARCHITECTURE

This section presents an overview of the proposed architecture for H.264/AVC encoder. The design of the architecture has been based on the analysis of the functionality of the encoder blocks and on their mapping on computing resources to produce the modules of the architecture. Figure 7 shows the hardware architecture of the inter and intra prediction, ICT/IICT and Q/IQ blocks. In fact, after loading the current MB and the reference search area, the decision block gives the residual MB which is obtained by subtracting the best predictor with the minimum SAD from the original. The residual MB is transformed using an integer transform, and the transform coefficients are quantized followed by zigzag ordering. The reconstruction process is applied in order to obtain a reference block for the next block.

The implementation technique of the different block composed our architecture will be detailled in next subsections.

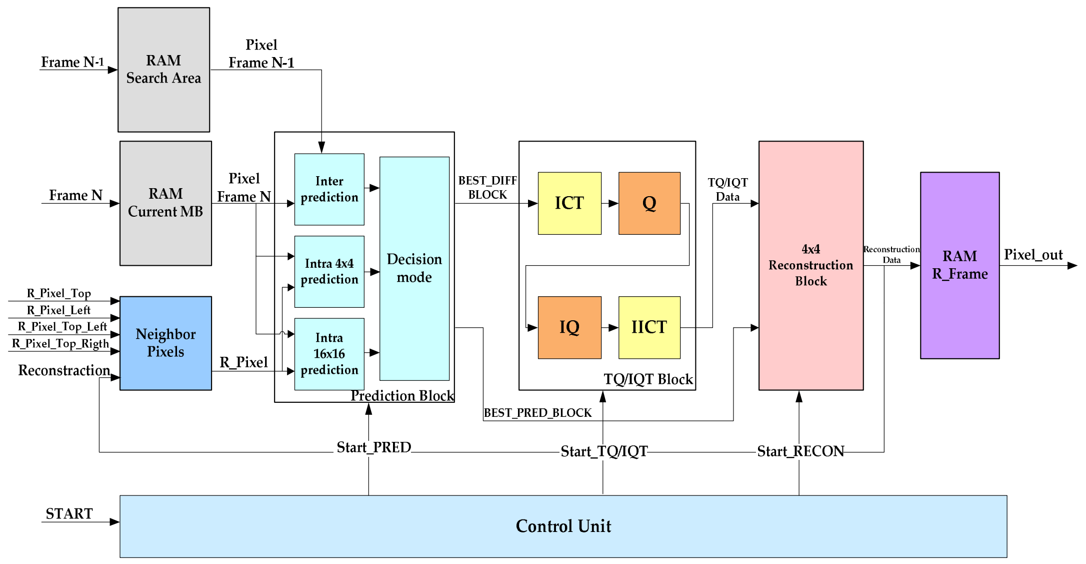


Figure 7. Hardware design for the H.264/AVC encoder

3.1. Proposed Prediction Hardware Architecture

The present section describes the details of the inter, intra 4x4 and intra 16x16 prediction blocks.

3.1.1. Inter Prediction Architecture

The hardware component of the LDPS algorithm is shown in Figure 8. This hardware architecture is composed of four sub-modules. The control unit is responsible for synchronization between different blocks of the search module, the extraction module, the SAD module and the comparator module in order to find the suitable MB for the reference MB in the defined search area.

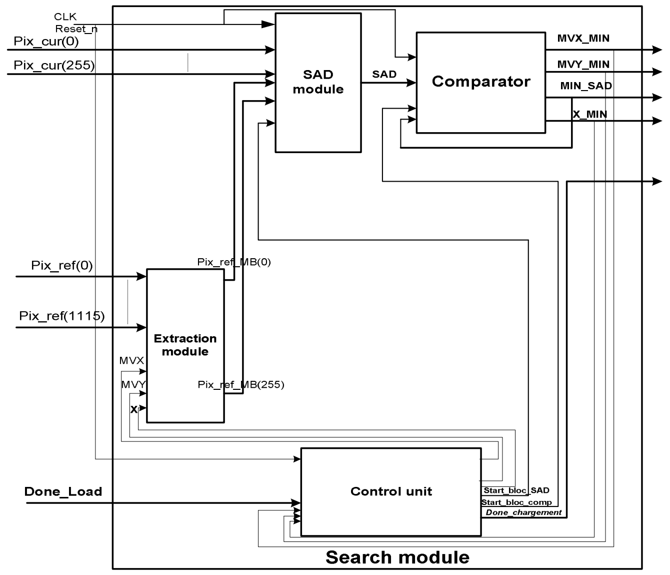


Figure 8. Block diagram of the LDPS architecture

In the first step, the extraction module extract pixels associated of the each reference MB through the search area memory per sending the specific address. This address is generated by the control unit by indicating the position of the small diamond or the line search as shown in Figure 9.

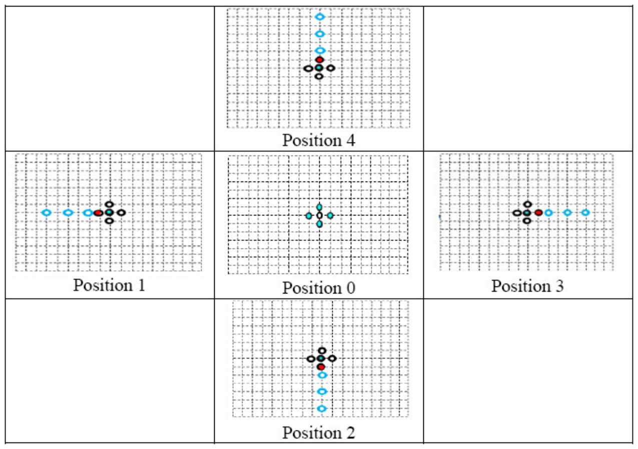


Figure 9. Position of the reference Macroblock

The second step allows evaluating SAD for the reference MB. This is why we have injected the appropriate reference MB in the SAD module (concerning the small diamond or the line search necessary to the LDPS algorithm). As illustrated in Figure 10, the SAD module is used to compute the nine SAD values such as one SAD 16x16, two SAD 16x8, two SAD 8x16 and four SAD 8x8. In fact, the pixels of the current and the reference MB are applied simultaneously to the SAD module through the “Pix_cur” and the “Pix_ref_MB” signals. Furthermore, the SAD module computes the difference, the absolute and the addition between the various pixels in sixteen clock cycle for the nine SAD.

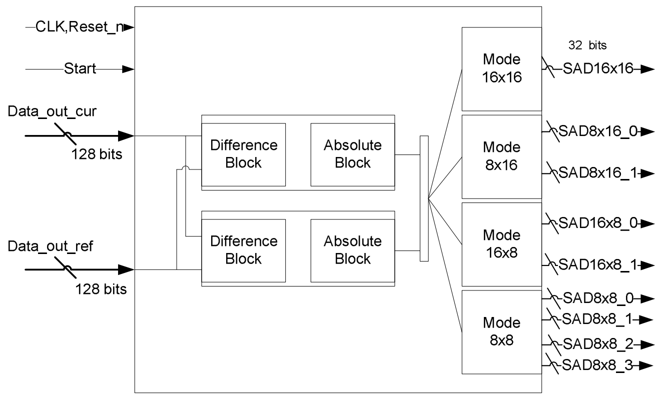


Figure 10. SAD module

After calculating the nine SAD values for one MB position in the search area, the comparator module, as shown in Figure 11, allows to accumulate the various SAD following the block size in order to obtain the best SAD 16x16. In fact, the comparator module accumulates four SAD 8x8, two SAD 16x8 and two SAD 8x16 to obtain one SAD 16x16 for each case and compare the various SAD values to select the best reference MB.

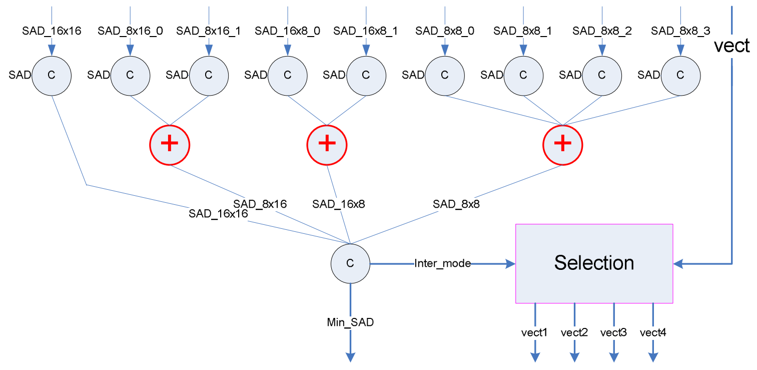


Figure 11. Comparator module

The LDPS algorithm is a non constant time algorithm. In fact, it provides the motion vector when the minimum SAD coincides with the current minimum position or the center of SDSP, as shown in Figure 3, or exceeding the search area. Therefore, the needed time to obtain the motion vector varies with the MB. For example, in the SDSP, the minimum SAD value needs 105 cycles where each position requires one clock cycle for read memory and extraction, sixteen clock cycles for computing SAD with variable block size and tree cycle for comparison. In fact, (21xN) clock cycles are needed to compute the motion vector for the best reference MB where N presents the number of the reference MB used to verify the stop criterions of the LDPS algorithm with VBSME (Variable Block Size Motion Estimation). Indeed, from the realized simulation, we conclude that the average time to determinate the best motion vector for several MB is 250 clock cycles.

3.1.2. Intra 4x4 Prediction Architecture

The intra 4x4 prediction architecture is composed by the predictor module, SAD calculator module and comparator module as shown in Figure 12. In this figure, the controller module receives input control signal (*Clk, Reset, Start*) and generates all the internal control signals for each stage and the output control signals for the communication with other hardware block.

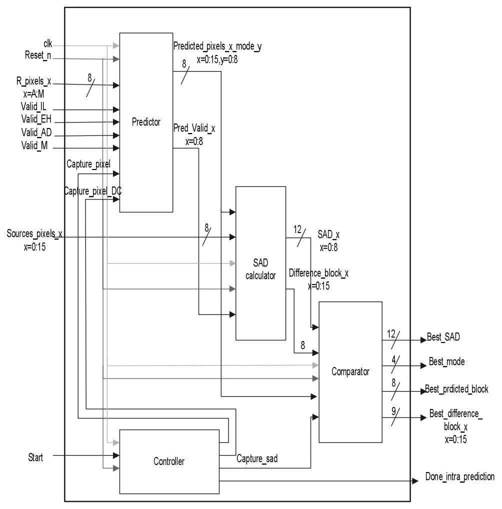


Figure 12. Intra prediction 4x4 design

The core of the predictor module is based on the prediction equations. In fact, after a careful analysis of the equations used in intra 4x4 prediction modes, it is observed that there are common parts in the equations and some of the equations are identical in each intra 4x4 prediction mode. The intra prediction equations are organized for exploiting these observations to reduce both the number of memory accesses and computation time required for generating the predicted pixels. Therefore, instead of breaking down the predictor calculation into individual calculator blocks for each mode, the calculation is divided into two stages: base and derived prediction equations as shown in Table 3 and Table 4 respectively.

Table 3. Base prediction equations

Output	Equation
Eq_0 =	A + B + 1
Eq_1 =	B + C + 1
Eq_2 =	C + D + 1
Eq_3 =	D + E + 1
Eq_4 =	E + F + 1
Eq_5 =	F + G + 1
Eq_6 =	G + H + 1
Eq_7 =	I + J + 1
Eq_8 =	J + K + 1
Eq_9 =	K + L + 1
Eq_10 =	M + A + 1
Eq_11 =	M + I + 1
Eq_12 =	2H + 1
Eq_13 =	2L + 1

Table 4. Derived prediction equations

Output	Equation	Derived Equations
Eq_14 =	M + 2*A + B + 2	Eq_0 + Eq_10
Eq_15 =	A + 2*B + C + 2	Eq_0 + Eq_1
Eq_16 =	B + 2*C + D + 2	Eq_1 + Eq_2
Eq_17 =	C + 2*D + E + 2	Eq_2 + Eq_3
Eq_18 =	D + 2*E + F + 2	Eq_3 + Eq_4
Eq_19 =	E + 2*F + G + 2	Eq_4 + Eq_5
Eq_20 =	F + 2*G + H + 2	Eq_5 + Eq_6
Eq_21 =	M + 2*I + J + 2	Eq_7 + Eq_11
Eq_22 =	I + 2*J + K + 2	Eq_7 + Eq_8
Eq_23 =	J + 2*K + L + 2	Eq_8 + Eq_9
Eq_24 =	A + 2*M + I + 2	Eq_10 + Eq_11
Eq_25 =	G + 3*H + 2	Eq_6 + Eq_12
Eq_26 =	K + 3*L + 2	Eq_9 + Eq_13
Eq_27 =	A + B + C + D + 2	Eq_0 + Eq_2
Eq_28 =	I + J + K + L + 2	Eq_7 + Eq_9
Eq_29 =	A+B+C+D+I+J+K+L+4	Eq_27 + Eq_28

The predictor module receives the thirteen neighbouring pixels from reconstructed blocks. Since not all of the neighbouring pixels may be available due to MB edges, there are valid inputs for each group of neighbouring pixels. The prediction calculator captures all reconstructed pixels and then calculates the equations needed to create all predicted values for all nine modes in parallel. The final operation is to fan-out the equation results to predicted values as shown in Table 5. The prediction calculator needs one clock cycle to generate all predicted pixels. The prediction calculator also creates a prediction valid (pred_valid_n) flag for each prediction mode.

The SAD calculator calculates the difference between the source pixels and the predicted pixels for all nine prediction modes in parallel. This operation needs 2 clock cycles. Finally, the SAD Comparator takes the 9 input SAD values corresponding to the 9 prediction modes and determines which SAD value is lowest. The lowest SAD value and its corresponding prediction mode are both output. This operation needs 2 clock cycles.

Table 5. Equation to Predicted Pixels Routing

Pixel	**Mode**								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
a	A	I	Eq_29	Eq_15	Eq_24	Eq_10	Eq_11	Eq_0	Eq_7
b	B	I	Eq_29	Eq_16	Eq_14	Eq_0	Eq_24	Eq_1	Eq_22
c	C	I	Eq_29	Eq_17	Eq_15	Eq_1	Eq_14	Eq_2	Eq_8
d	D	I	Eq_29	Eq_18	Eq_16	Eq_2	Eq_15	Eq_3	Eq_23
e	A	J	Eq_29	Eq_16	Eq_21	Eq_24	Eq_7	Eq_15	Eq_8
f	B	J	Eq_29	Eq_17	Eq_24	Eq_14	Eq_21	Eq_16	Eq_23
g	C	J	Eq_29	Eq_18	Eq_14	Eq_15	Eq_11	Eq_17	Eq_9
h	D	J	Eq_29	Eq_19	Eq_15	Eq_16	Eq_24	Eq_18	Eq_26
i	A	K	Eq_29	Eq_17	Eq_22	Eq_21	Eq_8	Eq_1	Eq_9
j	B	K	Eq_29	Eq_18	Eq_21	Eq_10	Eq_22	Eq_2	Eq_26
k	C	K	Eq_29	Eq_19	Eq_24	Eq_0	Eq_7	Eq_3	L
l	D	K	Eq_29	Eq_20	Eq_14	Eq_1	Eq_21	Eq_4	L
m	A	L	Eq_29	Eq_18	Eq_23	Eq_22	Eq_9	Eq_16	L
n	B	L	Eq_29	Eq_19	Eq_22	Eq_24	Eq_23	Eq_17	L
o	C	L	Eq_29	Eq_20	Eq_21	Eq_14	Eq_8	Eq_18	L
p	D	L	Eq_29	Eq_25	Eq_24	Eq_15	Eq_22	Eq_19	L

3.1.3. Intra 16x16 Prediction Architecture

Referring to the Figure 13, our Intra 16x16 prediction architecture calculates in parallel the predicted MB for all 3 intra 16x16 prediction modes specified in the H.264 standard (horizontal, vertical and DC) based on the reconstituted pixels from the previous MB (planar mode is not used [10]).

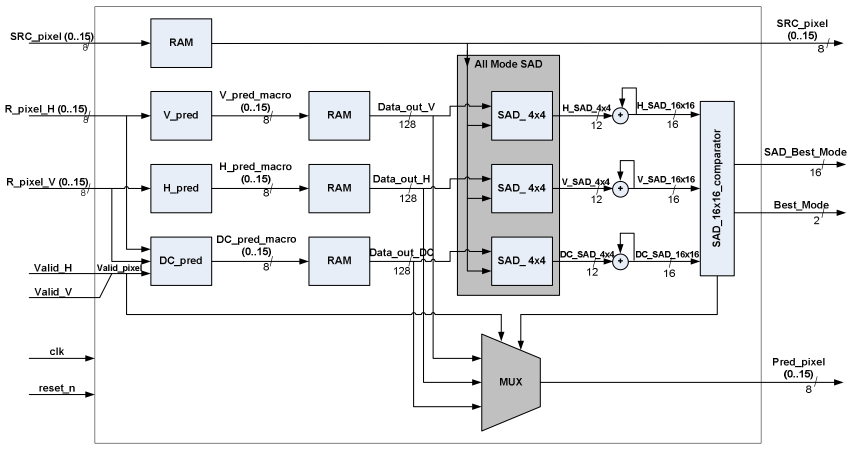


Figure 13. Intra prediction 16x16 design

In fact, the MB pixels are loaded into a dual RAM (Random Access Memory) for reordering and then output (to the residual or reconstruction blocks) by sets of 16 pixels (4x4 block). The predicted pixels are stored into RAM for all modes. We use a SAD_4x4 block for calculating the SAD value for each mode. We accumulate this value 16 times in order to obtain the SAD_16x16 for each mode. The comparator compares the SAD values for all prediction modes and picks the lowest SAD value for the best prediction mode. Indeed, from the realized simulation, we conclude that 108 clock cycles are necessary to obtain the best prediction mode whose 16 clock cycles for predictor block, 5 clock cycles to compute the SAD value for each 4x4 block and 2 clock cycles for the comparator block.

3.2. 4x4 ICT/IICT Hardware Architecture

The proposed architecture for the 2-D integer transform uses 4x4 parallel input data. A block diagram of this architecture is shown in Figure 14. This diagram contains two 1-D integer transform units and a control unit that provides clocks and others control signals such as the *Done_ICT* output flag signal to indicate that outputs coefficients are valid.

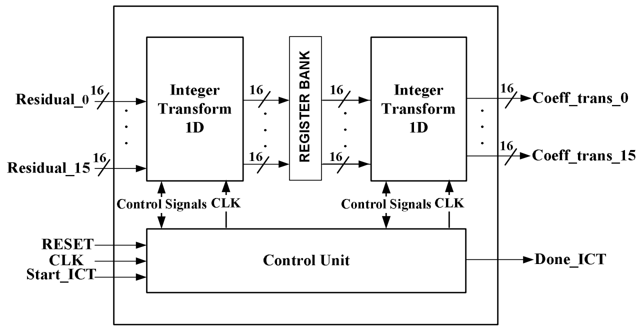


Figure 14. Architecture of the 2-D integer transform

In Figure 14, the 16 x 16-bit residual inputs data of the transform is captured from the outside environment through *residual_0..15* signal. Moreover, after intra or motion estimation prediction, the dynamic range of the inputs data is 9 bits, i.e. from -256 to +255. Because we have used operations like additions, subtractions and shifts, the dynamic range of the pixel data is extended to a 16-bit value [11]. So, the 4x4 residual data are processed in parallel by the transform block. This block consists of two cascaded 1-D transform units, i.e. one 1-D row transform and one 1-D column transform. The separable nature of the 2-D transform given by (3) is exploited by computing the 1-D transform on the rows and then the 1-D transform on the columns.

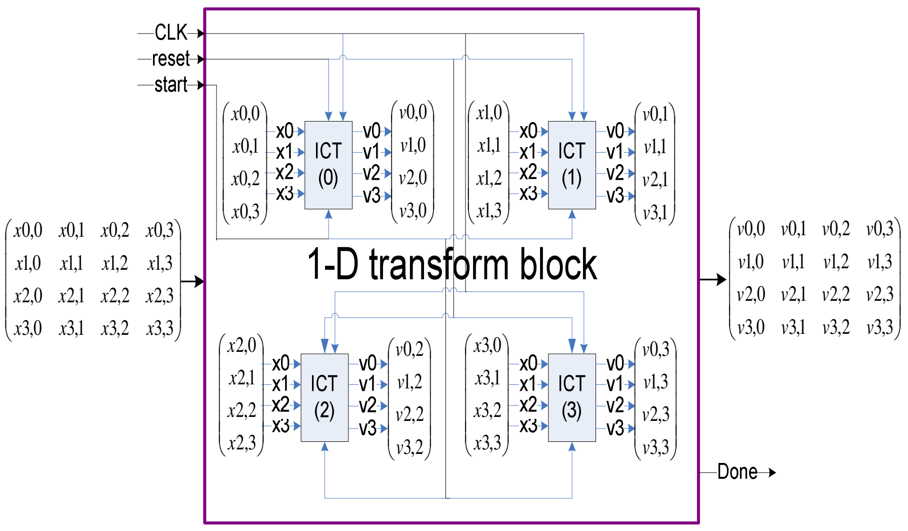


Figure 15. Architecture of the 1-D integer transform

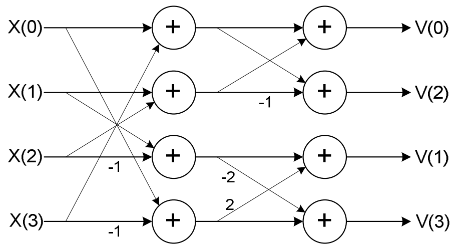


Figure 16. Fast implementation of 4x4 1D- ICT

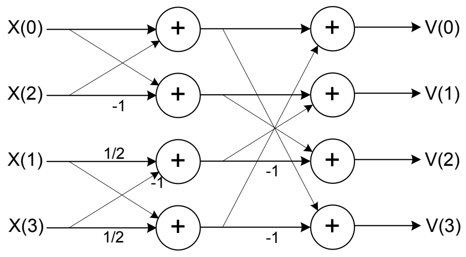


Figure 17. Fast implementation of 4x4 1D-IICT

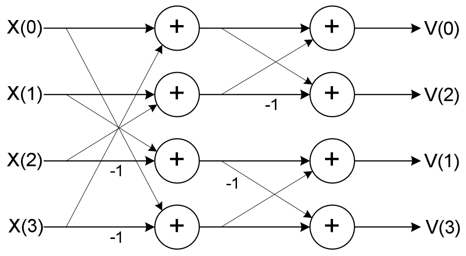


Figure 18. Fast implementation of 4x4 1D-HT

The parallel hardware architecture of the 1-D integer transform, 1-D ICT, unit is presented by Figure 15 and is designed to process 16 pixels per cycle by computing the transform of four lines in parallel. In fact, equation 17 is implemented by using concurrently four fast data-flow algorithm as detailed in Figure 16. Figure 17 and 18 are based on the same approach to present the fast implementation of the 1-D IICT and 1-D HT, respectively. These fast algorithms use only addition, subtraction and shift operations.

3.3. 4x4 Q/IQ Hardware Architecture

The proposed architecture for 4x4 AC and DC quantization is shown in Figure 19.

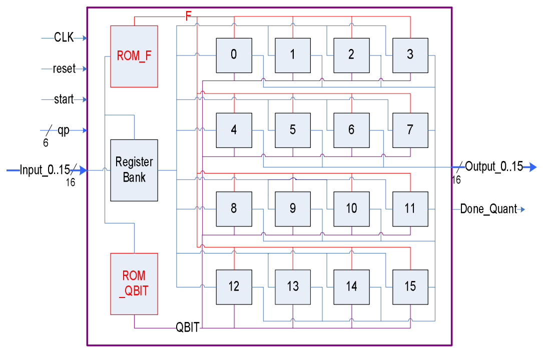


Figure 19. Architecture of AC and DC quantization module

The hardware quantization components for the AC and DC coefficients rescale the transformed coefficients according to the quantization step as defined by (12) and (14). It contains sixteen Processing Elements (PE), the register bank for storing the input pixels noted *input_0..15* and two read only memories (ROM) for storing QBIT and F values noted ROM_F and ROM_QBIT, respectively. The AC and DC quantization modules receive the sixteen 16 bits transformed coefficients in the same time and quantize these coefficients according to the QP factor in four clock cycles. The main component of the quantization architecture is the PE which shown in Figure 20. It is composed by four basic components and a control unit and is designed to quantize one transformed coefficient every four clock cycles. An integer multiplier assures the multiplication of AC and DC transformed coefficients with the corresponding MF(i,j) factor that is stored into the ROM_MF memory as shown in Table 2 and selected according to the QP modulo 6 value. The adder makes the sum of value given by the multiplier with the F parameter given by the ROM_F memory. A shifter register shifts the result set by the adder by *qbits* (varies 15 to 23 according to the value of QP). The multiplier, the adder, the shifter and the ROM_MF memory modules take one clock cycle each one. The control unit receives input control signals (*Reset*, *Clk*, *Start_Quant*) and generates all internal control signals for each stage and the output flag (*Done_Quant*) signal to indicate that the quantized coefficient is valid.

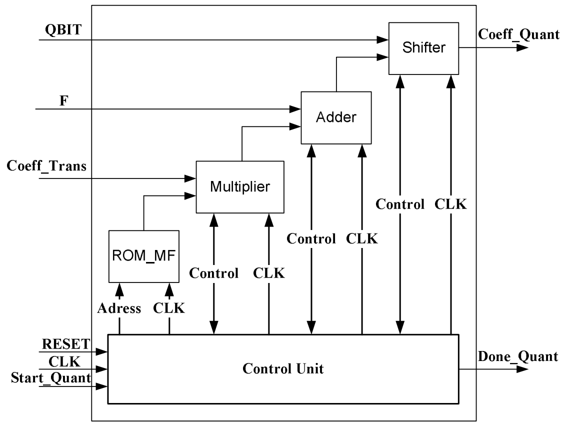


Figure 20. PE module of the quantization architecture

The inverse AC and DC quantization components share the same architecture design with AC and DC quantization presented in Figure 19. The differences between the architecture for the quantization and for the inverse quantization are presented in the PE module. In fact, for computing the inverse AC quantization values respecting (13), we have just eliminated the addition block from the PE module depicted according to Figure 20. On the contrary, to implement (15) the inverse DC quantization, we use the same PE module of the DC quantization. But the shifter block is implicated when QP<12. The AC and DC inverse quantization architecture is designed to provide sixteen coefficients every three and four clock cycles, respectively.

4. PERFORMANCE RESULTS

The proposed architecture is implemented in VHDL language. The implementation is verified with RTL simulations using Mentor Graphics ModelSim. It is then synthesized and placed and routed to a Stratix III EP3SL150 FPGA [12] using Altera Quartus II tool. Table 6 shows the hardware cost in terms of ALUTs (Adaptive Look-Up Tables), DSP (Digital signal processing) blocks which is introduced by Altera for signal processing applications [13] and RAM blocks.

Table 6. FPGA implementation of Hardware design for the H.264/AVC encoder

	ALUTs	DSP	RAM (Kbit)	Frequency (MHz)
Inter	4849 (4%)	0%	127 (<1%)	136
Intra 4x4	4358 (4%)	0%	0%	310
Intra 16x16	1951 (2%)	0%	0%	260
ICT	1024 (<1%)	0%	0%	480
IICT	1504 (<1%)	0%	0%	405
HT	1088 (<1%)	0%	0%	418
IHT	1056 (<1%)	0%	0%	436
ACQ	4108 (4%)	32 (8%)	6 (<1%)	221
IACQ	2013 (2%)	32 (8%)	2 (<1%)	269
DCQ	4047 (4%)	32 (8%)	6 (<1%)	211
IDCQ	5426 (5%)	32 (8%)	2 (<1%)	230
H.264 Design	**37178 (33%)**	**128 (54%)**	**150 (<1%)**	**130**

The entire hardware architecture for the H.264/AVC encoder uses 33% of the ALUTs, 1% of the RAM blocks, 54% of the DSP blocks and 10% of the IOBs. We can see that there is enough free space to add other H.264/AVC blocks. Our architecture operates at 130 MHz thus 7.7 ns delay for each coded data is required.

Figure 21, 22 and 23 present the performance of the proposed H.264/AVC FPGA architecture in term of the number of hardware cycles. This performance is evaluated with different types of the video formats indicating the time needed for processing one MB for each prediction mode (inter, intra 4x4, intra 16x16). The prototyping results are compared with the reference software results and the comparison confirms the correctness of the prototyped architecture.

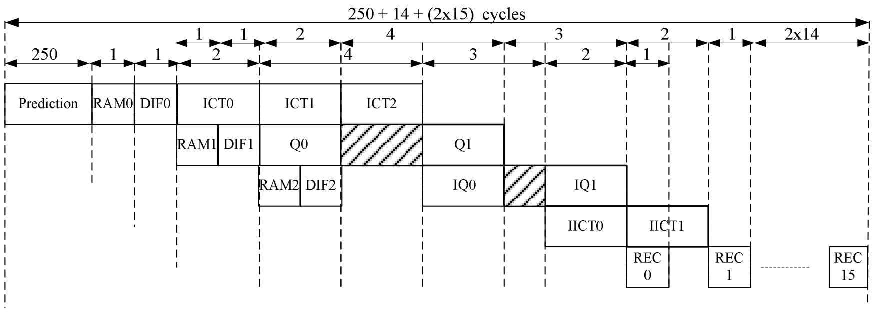


Figure 21. Hardware cycles for inter mode

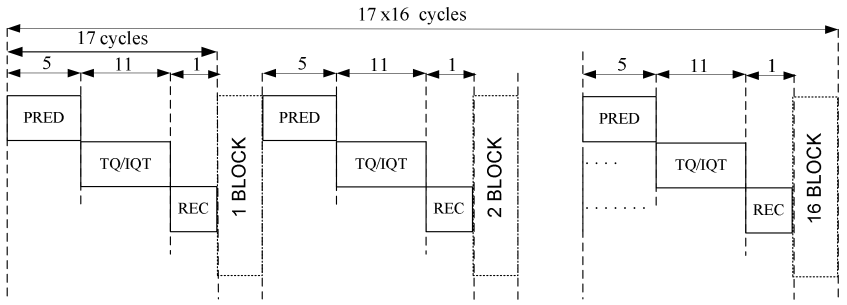


Figure 22. Hardware cycles for intra 4x4 mode

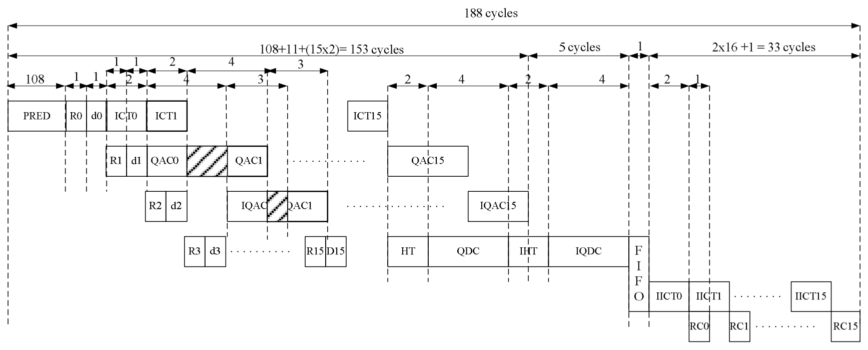


Figure 23. Hardware cycles for intra 16x16 mode

With operating clock frequency 130 MHz, the data throughput of the proposed architecture can achieve 113 Mpixels/sec, 122 Mpixels/sec and 177 Mpixels/sec which depend of the prediction mode inter, intra 4x4 and intra 16x16 respectively. The most important result is the maxima throughput of the internal H.264/AVC encoder architecture that, in all case, is sufficient to operate in H.264/AVC encoder for HDTV. Considering a HDTV 1080i (1920x1088@30Hz) video format and a downsampling relation of 4:2:0 then the required throughput is 94 Mpixels/s. The FPGA design of the H.264/AVC design is able in worse case, i.e., when the inter prediction mode is always chosen, to reach a processing rate of 113 Mpixels/s which is outperforming the HDTV requirement. So, aiming the target application, appropriate frequency can be chosen for the specific application in order to achieve lower power consumption.

5. CONCLUSION

This paper has presented an FPGA video encoder for H.264/AVC which achieves a real-time performance and has low area cost. Our parallel and pipeline design was described in VHDL and synthesized to the Altera Stratix III EP3SL150 FPGA can encode HDTV 1080i 30 fps video in real-time at 130 MHz.

As future works, it is planned to integrate in the design the entropy coding block, then other optimization can be performed and better results can be achieved.

REFERENCES

[1] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC.

[2] I. E. G. Richardson, “H.264 and MPEG 4 Video Compression-Video Coding for Next Generation Multimedia”, New York: Wiley, 2003.

[3] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the H.264/AVC Video Coding Standard”, *IEEE Trans. on Circuits and Systems for Video Technology* vol. 13, no. 7, pp. 560–576, July 2003.

[4] Imen Werda, Haithem Chaouch, Amine Samet, Mohamed Ali Ben Ayed, Nouri Masmoudi, “Optimal DSP-Based Motion Estimation Tools Implementation for H.264/AVC Baseline Encoder,” *IJCSNS International Journal of Computer Science and Network Security*, vol. 7, no. 5, 2007.

[5] Imen Werda, Haithem Chaouch, Amine Samet, Mohamed Ali Ben Ayed, Nouri Masmoudi, “Optimal DSP-Based integer Motion Estimation Implementation for H.264/AVC Baseline Encoder,” The *International Arab Journal of information Technology*, vol. 7, no. 1, January 2010.

[7] Tham Y J, Ranganath S, Ranganath M et al, “A novel unrestricted center-biased diamond search algorithm for block motion estimation,” *IEEE Trans. on Circuits and Systems for Video Technology*, vol.8, no. 4, pp 369-377, 1998.

[8] C. Zhu, X. Lin, and L.P. Chau, "Hexagon-based search pattern for fast block motion estimation", *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 12, no. 5, pp. 349–355, 2002.

[9] M. Gallant, G. Côté, F. Kossentini, "An Efficient Computation- Constrained Block-Based Motion Estimation Algorithm for Low Bit Rate Video Coding", *IEEE Trans. Image Processing*, vol. 8, no. 12, 1999.

[10] A. Kessentini, B. Kaanich, I. Werda, A. Samet and N. Masmoudi, "Low complexity intra 16x16 prediction for H.264/AVC", *ICESCA'08*, May 2008, Tunisia.

[11] H.S. Malvar, A. Hallapuro, M. Karczewicz, L. Kerofsky, “Low-complexity Transform and Quantization in H.264/AVC”, *IEEE Trans. On Circuits and Systems Video Technology*, vol. 13, pp. 598–603, July. 2003.

[12] Altera Stratix III development platform http://www.altera.com/products/devkits/altera/kit-siii-host.html

[13] A. Ben Atitallah, P. Kadionik, F. Ghozzi, P. Nouel, N. Masmoudi, Ph. Marchegay “Optimization and implementation on FPGA of the DCT/IDCT algorithm”, *IEEE ICASSP '06*, Toulouse, France, 14-19 Mai 2006.

Authors

Ahmed Ben Atitallah received his Dipl.-Ing and MS degree in electronics from the National Engineering School of Sfax (ENIS) in 2002 and 2003, respectively and Ph.D. degree in electronics from IMS laboratory, University of Bordeaux1 in 2007. He is currently an assistant professor at Higher Institute of Electronic and Communication of Sfax (Tunisia). He is teaching Embedded System conception and System on Chip. His main research activities are focused on image and video signal processing, hardware implementation, embedded systems.

Hassen Loukil was born in Sfax, Tunisia, in 1979. He received electrical engineering degree from the National School of Engineering-Sfax (ENIS) in 2004. He received his M.S. and Ph.D. degrees in electronics engineering from Sfax National School of Engineering in 2005 and 2011 respectively. He is currently researcher in the Laboratory of Electronics and Information Technology and an assistant at the University of Sfax, Tunisia. His research interests include signal and image processing, hardware implementation using FPGA, embedded systems technology.

Nouri Masmoudi received electrical engineering degree from the Faculty of Sciences and Techniques - Sfax, Tunisia, in 1982, the DEA degree from the National Institute of Applied Sciences-Lyon and University Claude Bernard-Lyon, France in 1984. From 1986 to 1990, he prepared his thesis at the laboratory of Power Electronics (LEP) at the National School Engineering of Sfax (ENIS). He received his PhD degree from the National School Engineering of Tunis (ENIT), Tunisia in 1990. From 1990 to 2000, he was an assistant professor at the electrical engineering department -ENIS. Since 2000, he has been an associate professor and head of the group ‘Circuits and Systems’ in the Laboratory of Electronics and Information Technology. Currently, he is responsible for the Electronic Master Program at ENIS. His research activities have been devoted to several topics: Design, Telecommunication, Embedded systems and Information technology.