11

Video Compression from the Hardware Perspective

Grzegorz Pastuszak
Warsaw University of Technology
Poland

1. Introduction

Many advanced multimedia applications require image compression technology with ever higher compression ratios and better visual quality. The need for the real-time high-efficiency video compression usually involves the use of hardware accelerators. In general, the development of architectures mapped into integrated circuits allows simultaneous processing of various data. On the other hand, the hardware framework suffers from limitations on the algorithm flexibility due to timing dependencies coming from the designed dataflow. Thus, the development of efficient video codecs in integrated circuits should take into account the algorithm details of the video codec. The following sections address various aspects of the video-compression design at the hardware architecture level. Section 2 analyzes the video coding dataflow and the design efficiency regarding timing and resources. To illustrate challenges in the hardware design, Section 3 reviews architectures of main modules of the H.264/AVC hardware encoder. The implementation results are given in Section 4.

2. High-performance coding

The real-time performance means that the encoder (decoder) must process all input (produce all output) video frames/fields/macroblocks in a limited amount of time. The section analyzes the codec structures in terms of timing properties and resource consumptions.

2.1 Dataflow

Video systems for the compression of greyscale visual information operate on the three-dimensional signal. An additional dimension is added to index colour and auxiliary components. Colour components refer to one of some colour spaces such as RGB, YUV, and YCbCr.

The dataflow in the encoder of visual data is depicted in Fig. 1. A video encoder consists of four main functional parts related to temporal modelling, spatial modelling, quantization, and binary coding. Frame (or field) in a video sequence can be processed in two basic modes. The first is called INTRA and exploits only spatial modelling, as for images. The second is called INTER and uses both modelling parts.

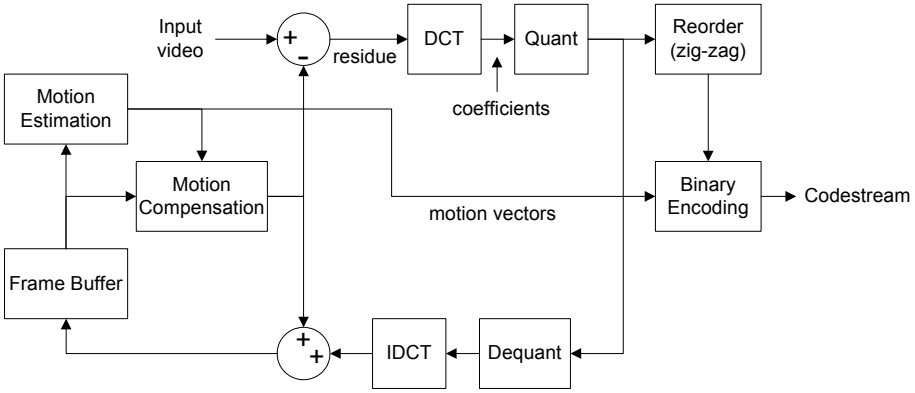

Fig. 1 Block diagram of the video encoder

The temporal model attempts to reduce temporal redundancy by exploiting similarities between neighbouring frames, usually by constructing a prediction of the current frame. The prediction is formed from one or more frames preceding or following the current one. When a selected reference frame is a previously encoded frame, the current one is referred to as a P-frame (see Fig. 2). When both a previously encoded frame and a future frame are chosen as reference frames, the current one is referred to as a B-frame. For a selected frame(s), the motion estimation (ME) module compares allowable pixel blocks (e.g., macroblocks) in the current frame with its surrounding area in the previous frame(s) and attempts to find the best match. The matching area (the prediction) is subtracted from the current macroblock in the motion compensation module. The difference between positions in the current and referred frames identifies motion vectors (MVs). If the motion estimation and compensation process is efficient, the remaining residual data should contain only a small amount of information. The temporal model outputs a residual frame and a set of parameters, typically the set of motion vectors.

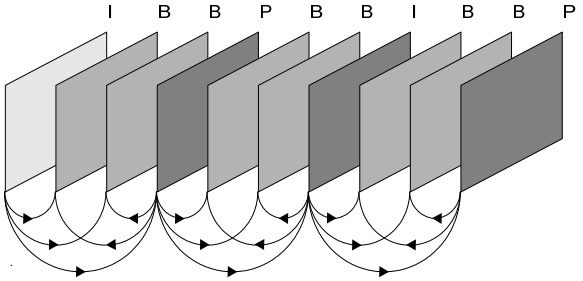

Fig. 2. I/P/B frames in a video sequence.

The spatial model exploits correlations between neighbouring samples within one frame to reduce spatial redundancy. This can be achieved by applying transform and/or prediction. The transform converts the samples into another domain in which they are represented by spatial frequency coefficients. Typically, the transforms operate on a two-dimensional block

of pixels rather than on a one-dimensional signal. Their ability to concentrate the signal energy enables few coefficients to recreate a recognizable copy of the original block of pixels. Apart from transform techniques, the spatial redundancy can be reduced using the prediction from neighbouring pixels within the same frame (interpolation and extrapolation).
For a typical block of pixels, most of the coefficients produced by the transform are close to zero. The quantization reduces the precision of each coefficient so that the near-zero coefficients are set to zero and only a few significant non-zero coefficients are left. Note that the quantization removes less important information.
The I- and P-frames must be stored in the buffer to be used as references when the INTER frames are encoded. The content of frames buffered in the encoder should be identical to the content of frames buffered in the decoder. Therefore, instead of simply copying frames into the buffer at the encoder side, they undergo some operations as in the decoder. In particular, to create a reconstructed frame, the quantized coefficients are rescaled, inverse transformed, and added to the motion-compensated reference block. These operations make up the feedback loop in the encoder. When the INTER frame is encoded, the motion estimator uses frames stored in the buffer to determine the best matching area for motion compensation.
The last step in the video coding process is binary coding that produces the output codestream. Inputs to the binary coder include transform coefficients for the residual data, motion vectors, frame pointers, block sizes, and other control information. The variety of these parameters, correlations between them, and their statistics affect the algorithm of binary coding, especially its complexity. The algorithm can adopt one or more coding methods. Finally, the type of binary coding depends on the application.

2.2 Timing

The section will analyze the number of clock cycles the codec can allocate to pixel-domain coding units. Moreover, the codec structure will be related the processing latency.

Pixel resolution	Time resolution	Throughput [MB/sec]	Max clock cycles per MB
576x720	25	40600	2461
480x640	30	36000	2777
720x1280	25	90000	1111
720x1280	30	108000	925
1080x1920	25	204000	490
1080x1920	30	244000	408

Table 1. Summary of timing requirements for different video formats

In order to satisfy real time requirements, the encoder throughput should be high enough. In practice, the required throughput depends on the video resolution related to time and pixel domains. They are measured in frames per second (fps) and pixel area, respectively. Additionally, subsampling of chroma components can affect the performance. As the video compression processes pixels in 16x16 pixel macroblocks, it is convenient to use the number of macroblocks per second to specify the throughput. Having a specified architecture, the performance depends on the clock frequency. In particular, the throughput is proportional to the frequency. Table 1 shows average macroblock throughputs required for different

resolutions and the average number of clock cycles allocated to each macroblock at 100 MHz. In practice, the hardware encoder performance should have a computation margin to compensate for wait states caused by initializations (e.g. probability models, rate control), the fullness of the output stream buffer, etc.

Apart from clocking the video codec core, it is important to provide the sufficient bandwidth to the external memory used to buffer original and reference frames. Particularly, each macroblock involves read access to one 16x16 original pixel block and some (N+5)x(M+5) reference pixel blocks. Note that N and M are the horizontal and vertical sizes of the reference area, respectively. The increase by five is the overlap which results from the subpixel interpolation. It is possible that the codec accesses to some smaller reference areas when a macroblock is partitioned and the partitions have different motion vectors and/or reference frames. Using more reference frames proportionally increases the number of read accesses to the reference area. As each reconstructed macroblock must be stored in this area, one 16x16 write access is performed for a macroblock. At the encoder side, input pixels should be stored in the external memory prior to reading original macroblock pixels, whereas the reconstructed frames are read and formed into output pixel stream at the decoder side. Thus, both sides need similar bandwidth to provide a pixel interface. If the bandwidth is not wide enough, the codec can encounter wait states decreasing its performance. In order to optimize communication with the external memory, one must employ efficient access scheduling between multiple write and read ports.

The video codec latency comes mainly from buffering input and output streams. In the encoder, the input pixel stream must be first stored in the memory line by line. If the number of pixel lines is sufficient to form 16x16 macroblocks, read access can start. In the case of emerging H.265 video standard, the traditional processing based on macroblocks is generalized to larger-size coding units (16x16, 32x32, and 64x64). As a consequence, the required number of buffered pixel lines increases accordingly. If the latency is not crucial parameter, the input buffer can keep more frames, i.e., the delay between writing and reading of the same pixels can be significant. In contrary to pixel streams, the amount of data in the code streams varies in time. Apart, from the bit-rate instability, transmission conditions change. When the bandwidth of the transmission channel between the encoder and decoder is limited, the buffer fullness also varies in terms of the amount of both code-stream and corresponding-pixel data. As the decoder buffer can underflow, the delay between decoding and displaying should be set to avoid situations when there are no decoded pixels to display in the output buffer.

Efficient hardware video codecs exploits the macroblock-level pipeline. The pipeline stages are distinguished with reference to mutual dependencies of processing blocks. In practice, the encoder embeds at least three stages associated with the motion estimation, internal loop (intra prediction, transforms, quantization, and reconstruction), and entropy coding in parallel with the deblocking filter. In the decoder, it is enough to exploit two macroblock-level stages since the motion estimation is not present.

The internal loop in the encoder involves some computation cycles for each macroblock when the Intra mode is analyzed. Particularly, the prediction for Intra 4x4 and 8x8 blocks is computed with reference to reconstructed pixels of blocks adjacent to the current one to the top and left side. Therefore, the processing of a block of the same size in the in the loop can start when the reconstruction for the top and left neighbours is finished. Owing to the number of blocks within the macroblock, the total number of clock cycles sacrificed to the

Intra 4x4 mode is equal to 16xN in the straightforward approach. N denotes the number of clock cycles between starting the prediction and finishing the reconstruction. Computations for other Intra and Inter (chroma/luma) modes can be interlaced with those for the Intra 4x4 blocks to reduce the number of clock cycles. This schedule does not have to decrease the total throughput as there are usually significant time gaps within all N-clock periods. Moreover, it is possible to schedule the processing so that some pairs of Intra 4x4 blocks can be computed immediately one by one without waiting for the reconstruction, i.e., reconstructions do not affect each other (Roszkowski & Pastuszak, 2010).

2.3 Resources

The section will review practical limitations on the amount of resources in available technologies and relate them to the complexity of video codecs. In general, it is possible to design the dataflow with very-high throughputs. In practice, the design should minimize the resource consumption due to the cost of silicon area and power consumption. When Application Specific Integrated Circuits (ASIC) are taken into account, encoder architectures (with the Inter prediction) reported in scientific literature consumes above 500K gates (see Table 2). For the Intra encoders the resource consumption can be significantly reduced below 100K gates. Note that the gate unit is equivalent to the basic two-input NOR/NAND gate. Additionally, designs embed some on-chip memories used as buffers with relatively quick data access. On contrary to the ASIC technology, the Field Programmable Gate Array (FPGA) devices embed other logic units which group the functionality of several gates. However, a simple mapping between the number of gates and logic units is difficult as it depends on the design, synthesis tools, and specific technologies. Due to the amount of logic resources, only the designs limited to the Intra mode can be easily mapped to FPGA technologies. The decoders are much simpler as they do not embed mode selection algorithms (Roszkowski et al., 2010).

Design	technology	Gate count	On-chip Memory [bit]	Max clock frequency [MHz]	Clock cycle per macroblock	features
Y. W. Huang, et. All (2005)	TSMC 0.25 μm	85K	14336	54	1300	Baseline SDTV, Intra
Lin Y.-K., et. All (2009)	TSMC 0.13 μm	94.7K	14720	140	560	Baseline 1080p, Intra
Lin Y.-L. S et. All (2010)	TSMC 0.13 μm	1697K	87040	158	632	High 1080p, Inter SR: 64x64
Liu Z. et. All (2009)	0.18 μm	1140K	887193	200	672	Baseline 1080p, Inter SR: 196x128
Chen Y.-H. et. All (2009)	TSMC 0.13 μm	452K	138854	54	~1330	Baseline D1, Inter SR: 64x32

Table 2. Comparison of different architectures

Important modules coupled with the hardware video codec with the support for Inter pictures are the external memories. They are used to store reference pictures and buffer original pictures (in the encoder). For high resolutions, a wide data width should provide a sufficient bandwidth. In practice, it can be achieved using several DDR(1/2/3) memories,

where the address/control bus is common, and the data bus distributed between memory chips (to increase data width). The memories and associated connections occupy the board area. Furthermore, the coupling with the external memories requires the memory controller with the scheduler to support some different ports. In practice, the controller embeds some on-chip memories to provide burst data access. Although, these resources are not taken into account when comparing different designs, their area cost can be significant.

3. Architecture design

The multimedia compression employs the sequence of processing steps, and each of them must apply separate approaches to optimize performance and resource consumption. Firstly, each processing block operates on different type of data at input/output ports. Secondly, the type of an operation involves specific timing dependencies and requires specific amount of resources. Thirdly, the block-level pipeline should be balanced in terms of throughput to utilize maximally all hardware resources. In the area of integrated circuit design for video technologies, most efforts concentrate on the development of standardized codecs from MPEG and H.26x series. The latest standard H.264/AVC allows the best compression ratio at the cost of computationally-intensive algorithms. Following subsections describe main processing blocks in the developed H.264/AVC video codec. This review allows the identification some challenges when facing the vide compression in the hardware framework.

3.1 Motion estimation

Block diagram of the developed ME system is presented in Fig. 3. The system is composed of the motion vector generator, compensator, the bank of 64 memories (fine search area and original data), the coarse-level full-search (FS) estimator, the interpolator, and the external memory controller. The architecture employs two-level hierarchical ME procedure. Thus, at the first stage, the coarse FS module performs FS on the whole search area (SA) subsampled with 16:1 ratio. To reduce the noise influence on initial MV accuracy, each pixel of the coarse SA is obtained by averaging of 16 pixels of the reference frame (Jakubowski 2008). The search range of the coarse FS is [-64, 63] pixels at most in both horizontal and vertical direction. When the coarse FS is completed, the interpolator fetches fine 40x40 reference samples from the external memory and generates quarter-pel ones within [-8, 7] range in both directions around the initial MV obtained from the coarse FS. The interpolator accepts eight column-oriented samples in a clock cycle. Therefore, processing of one colour component takes at least 200 clock cycles. Since every eight samples at the input corresponds to 128 ones at the output, memory write ports work at the doubled clock frequency.

Samples generated by the interpolator are loaded into the Fine Search Area space in SRAM. Thus, any search point inside the fine SA can be checked instantly with quarter-pel accuracy using the same hardware as for integer-pel MVs. For the sake of limited resources, ordinary SAD is used for evaluation of sub-pel MVs instead of sum of absolute transformed differences which requires the Hadamard transform. When interpolated fine SA is available in the Fine Search Area SRAM, the MV generator can perform adaptive ME according to the Multi-Path-Search algorithm described in (Jakubowski 2008). The MV generator sends MVs to the memories to obtain predictions. Based on these predictions the compensator

calculates residua and SAD values. The MV generator can determine the next step of the adaptation algorithm with reference to SAD values.
The compensator architecture is based on the pipeline design. It operates on 8×8 partitions and employs a SAD tree with four pipeline stages to generate SADs for all partition modes. Original and reference data are transferred from the local memories with double clock rate in the alternating way. Thus, in a single cycle of master clock, 64 samples of original and reference 8×8 blocks are fed to the SAD tree. Hence, to obtain SAD for the whole 16×16 MB, four clock cycles are necessary. Since during SAD calculation the next MV can be processed, every four clock cycles a new MV can be sent to the compensator. With such a setup, it is particularly beneficial to send MVs in long series, since it reduces the average time of single MV processing and increases the hardware utilization (avoiding wait states). Apart from the inter prediction, the compensator computes residua for intra predictions, which are first written to memories using 16x16-sample port.

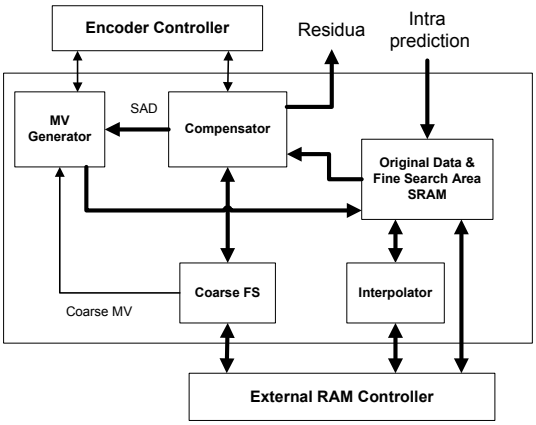

Fig. 3. Block diagram of motion estimator.

3.2 Intra prediction

High Profile of H.264/AVC standard defines three different kinds of INTRA prediction modes to be used for luma samples, and separate modes for chroma samples. Modes to be used for the luma sample prediction are: 4x4, 8x8, and 16x16, and are named after block sizes they operate on. The most commonly-used prediction modes are 4x4 ones. There are nine 4x4 modes, and eight of them are directional extrapolations of reconstructed samples from two neighbouring blocks (see Fig. 4). The ninth DC mode assigns the average of all reconstructed samples neighbouring with the current 4x4 block to predicted values. The 8x8 prediction modes are simple extensions of the 4x4 ones to blocks of the larger size. Therefore, there are also nine 8x8 modes, labelled identically as 4x4 ones. Except for the block size, the only difference comes from the prefiltering process. In particular, reference samples neighbouring with currently processed block undergo filtering before they are used for the prediction. Two of the directional modes: horizontal and vertical are the simplest since the prediction is equal to the copy of samples located to the left and above of the processed block, respectively. The remaining modes require some more complicated calculations according to the equations defined in H.264/AVC standard. Particularly,

predictors are determined using two simple equations where the result is the weighted average from two or three reference samples.

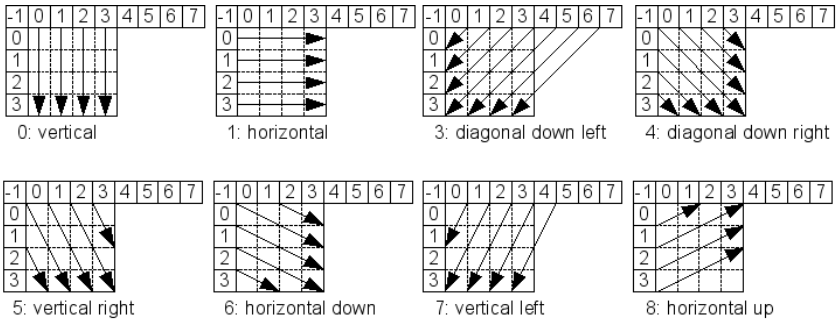

Fig. 4. Intra prediction modes for 4x4 blocks

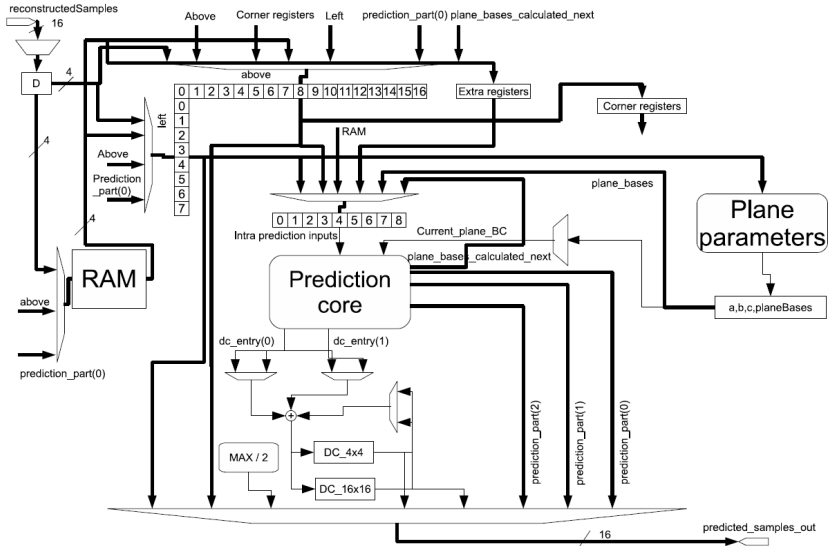

Fig. 5. Intra prediction block diagram

There are four 16x16 prediction modes defined by the H.264/AVC standard. Three of them: horizontal, vertical, and DC are simple extensions of corresponding 4x4 modes to 16x16 blocks. The fourth is the plane mode, the most computationally intensive one.

There are four chroma prediction modes defined for 4:2:0 and 4:2:2 sub-samplings. In fact, they are 16x16 luma prediction modes adapted to chroma block sizes. For 4:2:0 and 4:2:2, the prediction block size is 8x8 and 8x16 samples, respectively. In the case of 4:4:4 sampling scheme, there is no sub-sampling, and chroma predictions are obtained as the luma ones.

The Intra prediction architecture is described in details in (Roszkowski & Pastuszak, 2010). The architecture incorporates two important sub-modules that can be distinguished in the INTRA prediction module. These are: the neighbouring-sample buffer and the INTRA

prediction arithmetic core. The first sub-module is responsible for tracking which 4x4 block is to be processed next and the selection of neighbouring samples as the reference. The second sub-module calculates all prediction modes for the 4x4 block selected by the first one. Fig. 5 presents the neighbouring sample buffer sub-module. The most important part is the on-chip dual-port RAM module. It keeps reconstructed samples neighbouring with the currently processed macroblock and reconstructed samples inside the macroblock, which are needed to calculate the prediction for next 4x4 blocks. The raster order of macroblocks involves keeping the whole frame line in the RAM to provide adjacent samples from the top-neighbouring macroblock. Since both 4x4 and 8x8 predictions are computed in the interleaved manner, reconstructed samples for the two modes must be stored, which increases the memory space. Each memory cell keeps four adjacent samples.

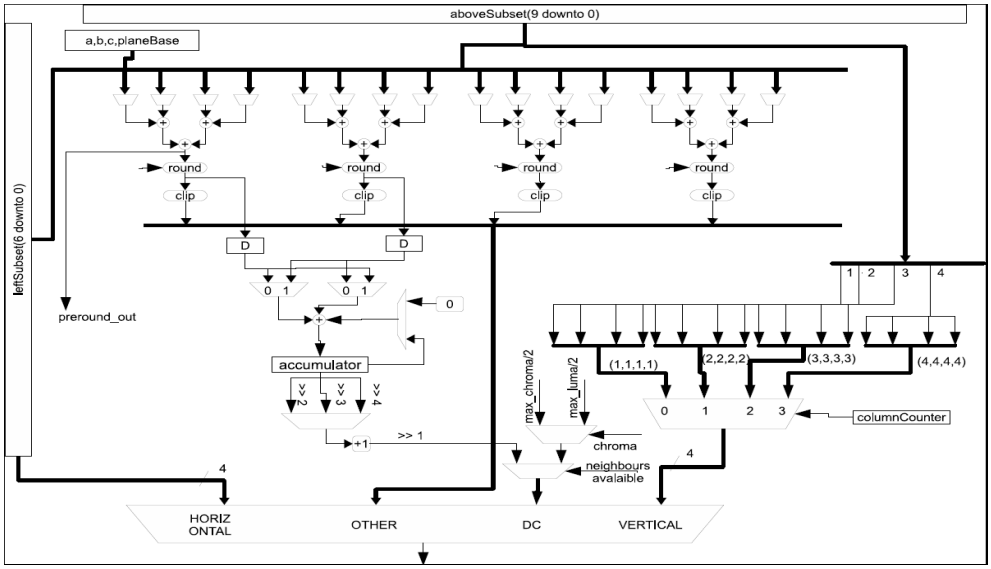

Fig. 6. Intra prediction arithmetic core

Plane prediction mode parameters are calculated in a separate sub-module in parallel with the calculation of 16x16 or chroma vertical and horizontal prediction modes. This allows a significant complexity reduction of the calculations of plane mode parameters as the multiplications can be replaced by the series of shift, addition, and accumulation operations. The input values to the prediction core are kept in nine intermediate registers. The rest of the module consists of the two levels of adders and multiplexers (see Fig. 6). The first and second levels of adders are responsible for the computation of the prediction values using three and two reference samples, respectively. As the result of the calculation, 15 different prediction values are obtained, out of which only up to 10 are valid for a 4x4 block. Those 10 are selected by the output multiplexer (MUX). The DC mode requires the reconfiguration of the adder structure, which is accomplished by multiplexers coloured dark grey in Fig. 7. The new configuration, together with the extra adder, allows the calculation of the prediction of the whole 4x4 block in one clock cycle. The prediction for 8x8 and 16x16 blocks, done by the

accumulation, takes 2 and 4 clock cycles, respectively. The remaining multiplexers are used to reconfigure the core for the plane prediction.

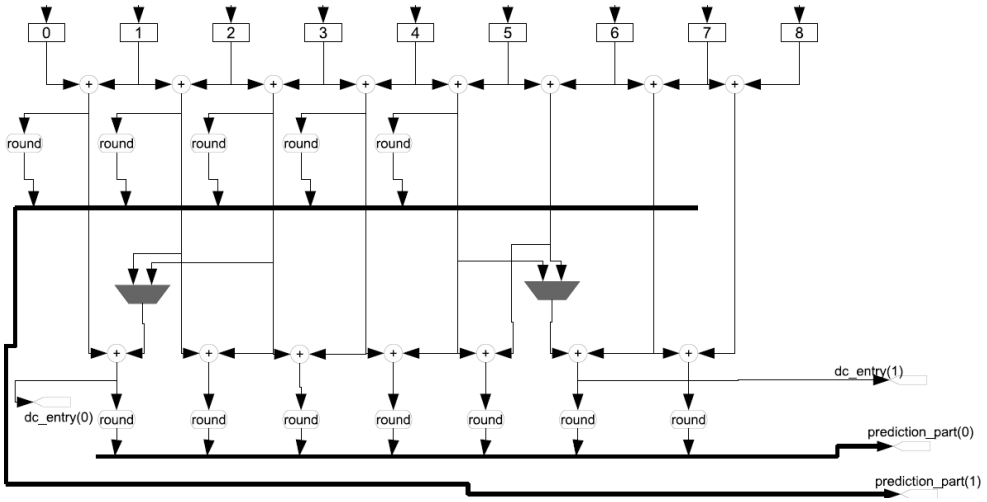

Fig. 7. Intra prediction modes for 4x4 blocks

3.3 Transforms

The primary transform applied in H.264/AVC is an exact-match integer 4×4 spatial block transform, which approximates DCT. The forward and inverse 4x4 transforms are shown in the Equation 1 and 2, respectively.

$$T_{FORWARD\_4x4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \tag{1}$$

$$T_{INVERSE\_4x4} = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \tag{2}$$

A secondary transform (Hadamard) performed on DC coefficients of the primary transform (for chroma DC coefficients and also luma in the 16x16 mode) allows for even more compression in smooth regions. Both transforms are similar, i.e., the secondary uses only 1 and -1 values in the matrix. For High Profile, the encoder can adaptively select between a 4×4 and 8×8 transform size for luma. The forward and inverse 8x8 transforms are shown in the Equation 3 and 4, respectively. As can be seen, the inverse matrix is a transposed version of the forward one.

$$T_{FORWARD\_8x8} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \frac{3}{2} & \frac{5}{4} & \frac{3}{4} & \frac{3}{8} & -\frac{3}{8} & -\frac{3}{4} & -\frac{5}{4} & -\frac{3}{2} \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 & -1 & -\frac{1}{2} & \frac{1}{2} & 1 \\ \frac{5}{4} & -\frac{3}{8} & -\frac{3}{2} & -\frac{3}{4} & \frac{3}{4} & \frac{3}{2} & \frac{3}{8} & -\frac{5}{4} \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ \frac{3}{4} & -\frac{3}{2} & \frac{3}{8} & \frac{5}{4} & -\frac{5}{4} & -\frac{3}{8} & \frac{3}{2} & -\frac{3}{4} \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} & -\frac{1}{2} & 1 & -1 & \frac{1}{2} \\ \frac{3}{8} & -\frac{3}{4} & \frac{5}{4} & -\frac{3}{2} & \frac{3}{2} & -\frac{5}{4} & \frac{3}{4} & -\frac{3}{8} \end{bmatrix} \tag{3}$$

$$T_{INVERSE\_8x8} = \begin{bmatrix} 1 & \frac{3}{2} & 1 & \frac{5}{4} & 1 & \frac{3}{4} & \frac{1}{2} & \frac{3}{8} \\ 1 & \frac{5}{4} & \frac{1}{2} & -\frac{3}{8} & -1 & -\frac{3}{2} & -1 & -\frac{3}{4} \\ 1 & \frac{3}{4} & -\frac{1}{2} & -\frac{3}{2} & -1 & \frac{3}{8} & 1 & \frac{5}{4} \\ 1 & \frac{3}{8} & -1 & -\frac{3}{4} & 1 & \frac{5}{4} & -\frac{1}{2} & -\frac{3}{2} \\ 1 & -\frac{3}{8} & -1 & \frac{3}{4} & 1 & -\frac{5}{4} & -\frac{1}{2} & \frac{3}{2} \\ 1 & -\frac{3}{4} & -\frac{1}{2} & \frac{3}{2} & -1 & -\frac{3}{8} & 1 & -\frac{5}{4} \\ 1 & -\frac{5}{4} & \frac{1}{2} & \frac{3}{8} & -1 & \frac{3}{2} & -1 & \frac{3}{4} \\ 1 & -\frac{3}{2} & 1 & -\frac{5}{4} & 1 & -\frac{3}{4} & \frac{1}{2} & -\frac{3}{8} \end{bmatrix} \tag{4}$$

To simplify computations, all the transforms should be decomposed into two or three stages using butterfly structures. Actually, the standard defines the inverse transforms in this form. Thus, rounding operations must be performed in the decomposed form to keep the specification consistency. For a block, appropriate matrix is applied on each row and then on each column to obtain the 2D transform.

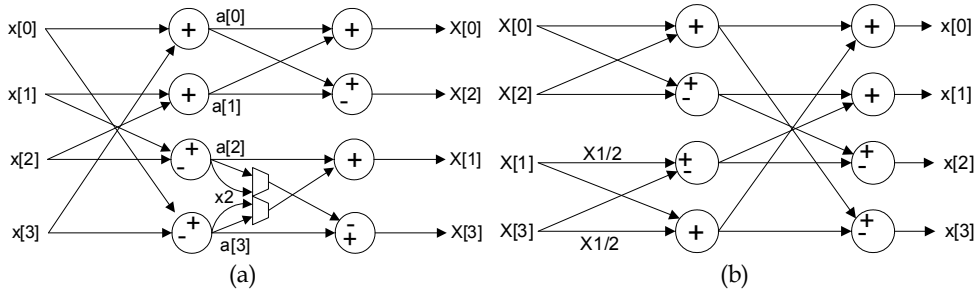

Fig. 8. Diagram of the forward and inverse transforms for 4x4 blocks

The best way to implement a transform is to use its decomposed form. Such a decomposed form is depicted in Fig. 8 for the 4x4 blocks and in Fig. 9 for 8x8 blocks. The forward 4x4 transform in Fig. 8.a supports both the approximate DCT and the Hadamard transform. Particularly, additional multiplexers enable a small reconfiguration of the connectivity. The transforms for 8x8 blocks are more complex. They consist of four processing stages, whereas two processing stages are used for 4x4 blocks.

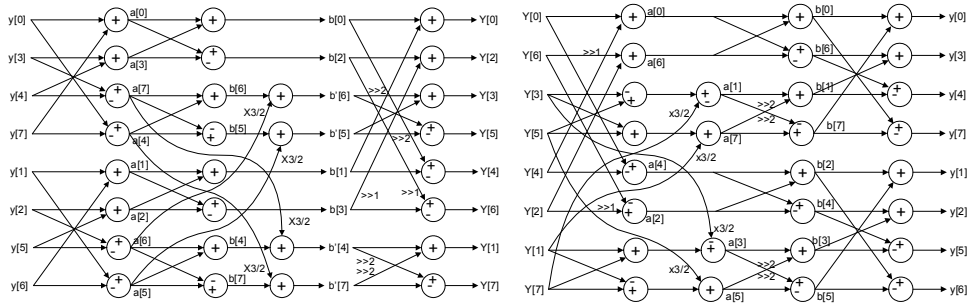

Fig. 9. Diagram of the forward and inverse transform for 8x8 blocks

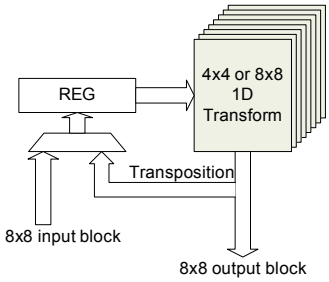

Fig. 10. Block diagram of the transform module

When multiple transforms are to be supported, the encoder can simply embed dedicated modules, each of which supports one transform type. To keep dataflow regularity in the forward or inverse transform, two modules for the four-element transform (4x4 blocks) and one for the eight-element transform (8x8 blocks) should be employed. The selection between two transform types is done by multiplexers placed at the output stage. Such a design is inefficient in terms of hardware resources since only one branch is used at one time. Thus, the efficient solution should utilize the same resources with the overhead as little as possible. The transform architecture with higher throughput can be easily designed by employing eight parallel eight-point transform logic units, as shown in Fig. 10. The result is computed in two clock cycles, 1D transform is performed in one cycle. More details about sharing resources between the two transform types can be found in (Pastuszak, 2008.a).

3.4 Quantization

The forward and inverse transform matrices are not orthogonal. To achieve this feature in the whole processing, quantization step sizes are modified. As a consequence, the step size depends on the position in the coefficient block. Actually, the quantization and dequantization are accomplished by the multiplication and shifting operations. Equations 5 and 6 show formulas for the quantization and the dequantization, respectively.

$$X_q(i,j) = sign\{X(i,j)\}(|X(i,j)|\,A(Qp\%6,i,j) + \tfrac{1}{3}2^{11+L+Qp/6}) >> (11+L+Qp/6) \qquad (5)$$

$$X_r(i,j) = (X_q(i,j)B(Qp\%6,i,j) + 2^{L-Qp/6}) << (Qp/6 - L) \quad (6)$$

In these equations, L is equal to either 4, 5, or 6 and depends on the transform size. Functions A and B include values of multiplicands for each location in the block. The values depend on the quantization parameter (Qp) and the transform size. Note that the position inputs identify the coefficient location in the rectangular structure (4x4 or 8x8). Block diagrams in Fig. 11 show dataflow of the quantizer and the dequantizer. In contrary to the dequantization, the quantizer embeds the addition of a fraction dependent on the coefficient sign. Since the units map one input into one output, it is easy to parallelize them to increase the throughput.

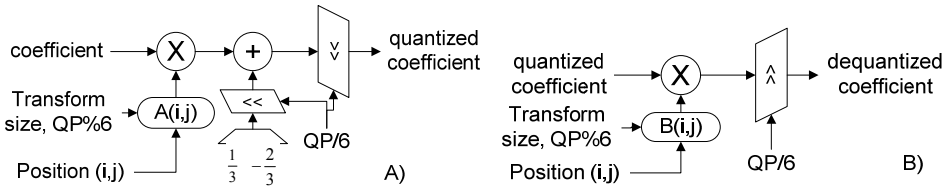

Fig. 11. Block diagram of the quantization (A) and dequantization (B)

3.5 Reconstruction

Following dequantization, reconstructed residuals are added to the prediction (intra or inter) to obtain reconstructed samples. As prediction samples in the encoder are computed in the motion estimation and intra prediction units, their bypassing to the reconstruction stage involves significant storage resources and write conflicts. An alternative is to refer to original residuals (registered at the transform input) and original samples in two successive clock cycles. In the first cycle, this approach allows the computation of the reconstruction error equal to the difference between original and reconstructed residuals. The reconstruction error subtracted from original samples gives reconstructed samples in the second cycle. In the high-throughput datapath, many subtractors can be utilized to perform the parallel reconstruction (see Fig. 12). To avoid underflow/overflow, the result is limited to the pixel sample range in the following pipeline stage.

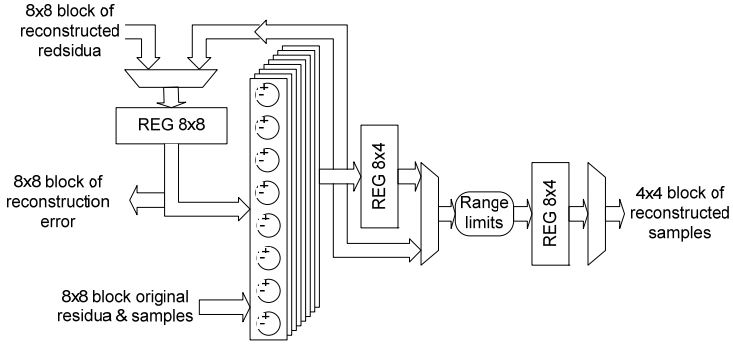

Fig. 12. Block diagram of the reconstruction unit

3.6 Mode selection

The simplest way to select the coding mode is to compute Sum of Absolute Differences (SAD) for each tested prediction and to select the case with the minimal SAD. This approach does not provide the optimal mode selection. A more advanced approach refers to actual code-stream rates and distortions. However, this involves much more computations and storage resources. The cost measure for a given mode is based on the cost functions, according to the following equations:

$$J_1(R,D) = D + R * \lambda \tag{7}$$

$$J_2(R,D) = D * \lambda^{-1} + R \tag{8}$$

Note that λ is the Lagrangian multiplier whose value is adjusted to the desired compression ratio. The J_1 and J_2 cost measures are expressed in distortion and rate domain, respectively. In the developed architecture the second measure has been selected as the multiplication is performed only once after obtaining the distortion. The distortion is computed based on the reconstruction error (see previous section). Particularly, the error for each sample should be squared, and the results can be summed within 4x4 partitions. Such Sum of Squared Errors (SSE) can be multiplied by the Lagrangian multiplier. As the developed architecture operates on 8x8 blocks, the distortions for four 4x4 subpartitions are summed, and only one multiplication circuit is enough for the assumed throughput of 32 samples/coefficients per clock cycle. The cost for larger partitions can be obtained by summing costs for smaller ones.
The second factor in the cost function is the rate measured in bit units. To estimate actual rates, the analysis of quantized coefficients following the binarization schemas is indispensible. There are two entropy coding modes with different schemas. When the CABAC is used, coefficients are coded using Exp-Golomb schema before arithmetic coding. Although the estimation of rates based on single coefficient values is easy, the probability adaptation can affect the estimation accuracy. On the other hand, the CAVLC binarization is based on the concatenation of successive codewords. Thus, the total rate is the sum of codeword rates. Since the CAVLC adapts binarization schemas while coding coefficients within 4x4 blocks, the estimation of coefficient rates involves the signal chain between 16 subcircuits corresponding to each coefficient. To shorten critical paths, the subcircuits should be placed in successive pipeline stages.
The block diagram of the mode selection module is depicted in Fig. 12. The parallelism employed in the developed encoder enables the repetition of quantization and transformation for different coding options to select the best one. In particular, it is assumed that the pipeline can process 8x8 blocks at the average throughput of 32 samples/coefficients per clock cycle. Hence, the module is able to check four 8x8, two 16x8, and two 8x16 partitions in successive eight clock cycles. The 16x16 partition (not partitioned macroblock) is analyzed in the separate path that simply aggregate costs of four successively analyzed 8x8 partitions. Addition of side cost (e.g., motion vectors, intra directions, macroblock/submacroblock types) allows a more reliable cost comparison. Actually, motion vectors and intra directions are coded using the prediction from the top and left neighbours. The dedicated memory (context) keeping picture line data allows the reference to the top neighbours, excluding cases when the reference partition belongs to the same macroblock.
As the mode selection for a macroblock and its partitions takes some time, it is necessary to buffer quantized coefficients for some different modes. When the macroblock mode is

selected, quantized coefficients comprising a 4x4 block are accessed concurrently at the entropy coder side, so that they are read in form 16x8-bit memory buffer. This parallel access results from the fact that such an order is at the write port. The analysis path uses pointers to identify addresses of 8x8 blocks stored in the buffer. Additionally, a vector of three-bit registers (kept in the write stage) identifies how many references to an 8x8 partition at a given address are valid. 8x8 partitions are written at four address identified by one pointer, and each address corresponding to a 4x4 block is distinguished by two bits based on its location. If intra and inter blocks are written, the corresponding register is set to one and four, respectively. If a reference is no longer valid, the pointers select which register should be decremented (discarded pointer). Four references match the case when an 8x8 partition contributes to the macroblock mode selection for four portioning types. Actually, each partition can have a different motion vector and reference picture selected based on the cost minimization. While the final macroblock mode is not selected, the best partition mode for both transform sizes and some quantization parameters should be looked for. This requires additional storage resources to save pointers, costs, motion vectors, and reference pictures (partition cost buffer and 16x16 cost buffer with pointers). Also intra modes should have storage space assigned. Taking into account the throughput, it can be seen that the analysis of partitions larger than 8x8 requires the pipeline registers carrying coding mode parameters and costs. This correspond to the first part (partition cost) of the mode selection block diagram in Fig. 13.

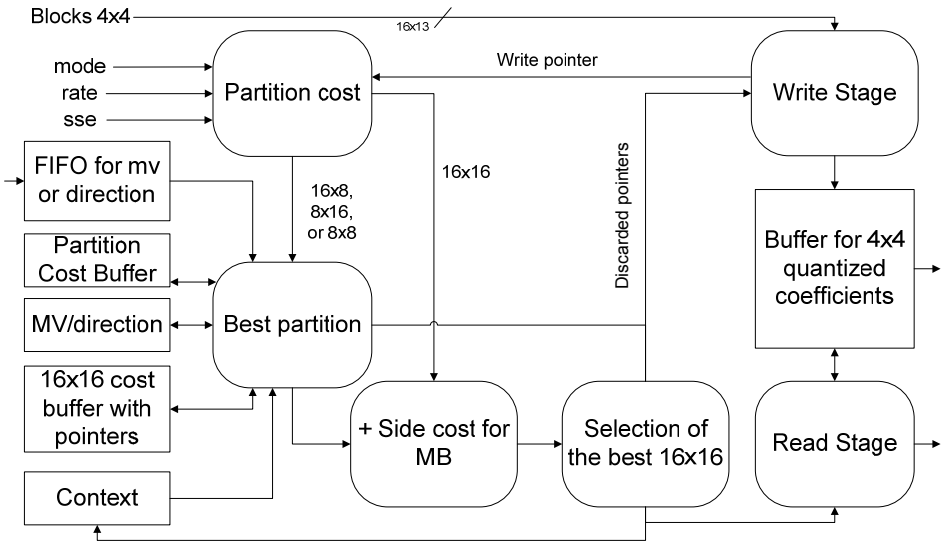

Fig. 13. Block diagram of the mode selection module

3.7 Entropy coding

In H.264/AVC two modes are employed for binary coding: Context Adaptive Binary Arithmetic Coding (CABAC) and Context Adaptive Variable Length Coding (CAVLC). The first mode provides higher compression efficiency at the cost of computational complexity.

The following subsections review the processing blocks for the two modes. More details can be found in the reference (Pastuszak, 2008.b).

3.7.1 Variable length coding

Since residual coefficients comprise the largest part of the codestream, exploiting correlations between them considerably improves the compression efficiency. Five types of syntax elements are processed in the CAVLC mode:

- For luma blocks, the total of non-zero coefficients and trailing ones (series of high-frequency coefficients equal to one) are coded as one element by the use of four look-up tables (three VLC tables and one 6-bit fixed table, each having 64 entries). The tables are selected adaptively based on the number of non-zero coefficients in the neighbouring 4x4 blocks. Besides, there are three additional tables for chroma blocks.
- Sign coding does not require context modelling, since one bit per non-zero coefficient is enough to convey this information.
- The code for each coefficient level is made up of a prefix and a suffix. The length of the latter one is initialized to either 0 or 1 and incremented every time when consecutive levels exceed predefined thresholds. This adaptation is due to the observation that statistically values of coefficients increase while passing from high to low frequencies.
- The total of zero-valued coefficients (total_zeros) preceding the last non-zero coefficient in the coding order refers to some VLC tables. One table is selected based on the number of non-zero coefficients coded earlier.
- The number of zeros preceding each non-zero coefficient (run_before) is encoded in reverse order. The adaptation is performed by the selection of codes dependent on the number of zero-valued coefficients left to be coded in this order.

The developed architecture of the H.264/AVC binary coder embeds the binarization unit as a part sufficient to support the CAVLC mode and perform the binarization in the CABAC mode. The binarization unit embeds four pipeline stages, as depicted in Fig. 14. Most of registers incorporated to the architecture are shared in both coding modes. Input data are submitted through dedicated ports, each of which matches one type of syntax element.

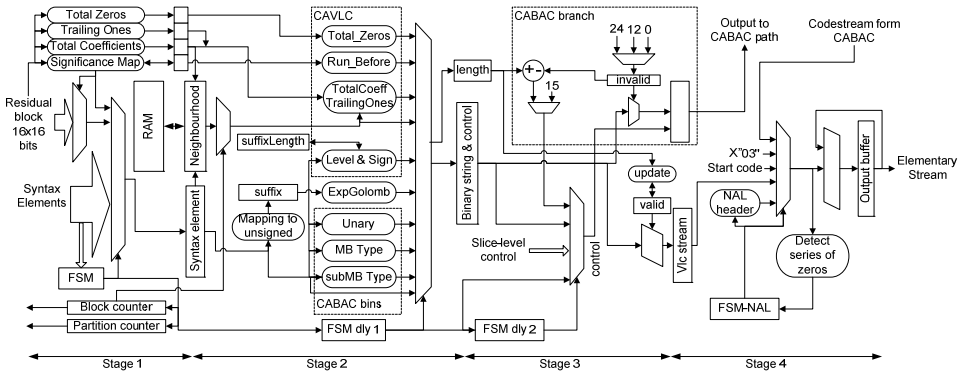

Fig. 14. Block diagram of the double-mode binarization unit

The binary coder processes syntax elements in the order defined in the standard. The order depends on selected options and previous data. Therefore, the architecture incorporates a

Finite State Machine (FSM) to determine the type and the order of the processed data. Transitions of the FSM depend on the values of syntax elements available on parallel input ports. The first stage selects one input port and loads corresponding data to the syntax-element buffer on the basis of the state of the FSM, counters, and a significance map. The FSM determines the type of the syntax element, whereas the counters point one subunit of a given macroblock such as a partition and a block (4x4). One FSM used in two modes simplify the design as states and transitions are almost the same. The main difference in transitions comes from the order of syntax elements within a 4x4 block. In the CAVLC mode, each block is scanned two times (i.e., non-zero coefficient levels precede runs of zero coefficients) whereas in the CABAC mode, just one scan is enough.

The first stage analyzes each 4x4 block to compute the number of non-zero (Total Coefficients) and zero-valued (Total Zeros) coefficients, the number of trailing ones, and the significance map. The significance map consists of 16 bits, where each bit is set active when the corresponding coefficient is non-zero. This allows the selection of coefficients to be processed. When a coefficient is selected, the corresponding significance indicator is set inactive. In the CAVLC mode, the first stage performs also the reference to a total of non-zero coefficients and trailing ones for the upper- and the left-neighbouring blocks. The referred numbers are used to compute the average (nC) forwarded to the second stage.

Raster scanning of macroblocks involves the use of an on-chip memory to convey references between rows. The memory incorporated into the architecture has the bit width equal to 48. This value matches the accumulated length of reference registers on one macroblock edge. The number of entries determines the maximal frame width in macroblocks and is set to 128 allowing HDTV resolutions.

Although context-formation rules for the CABAC differ from those for the CAVLC, it is possible to share storage elements in both modes. Thus, in the CABAC mode, the architecture keeps motion vectors differences instead of non-zero coefficients in the neighbourhood registers and the on-chip memory. However, the storage space is doubled since four six-bit MVD can be used for the smallest 4x4 partition. Nevertheless, sharing enables the efficient reduction of hardware resources. Additionally, the control subcircuit is common to both modes.

The second stage maps syntax elements onto their binary representation using the set of primitives (subcircuits) implemented as a combinational logic. Apart from a binary string, the primitives produce the corresponding length. For a given syntax element, a one-cycle delayed FSM selects the outcome of one primitive. The primitives support Unary, Exp-Golomb, macroblock, and submacroblock binarizations. The second stage includes dedicated subcircuits for adaptively-coded syntax elements in the CAVLC mode (i.e., 4x4 residual blocks).

The third stage forwards all code strings produced in the second stage to one of two paths. The first path, which supports the CABAC mode, assembles a binarized representation of a syntax element along with control data into 16-bit words and submits them to the context formatter. Each syntax element allocates bits in a specific way. The control information data includes the number of valid bits, indicators of the last syntax element in a series (e.g., coefficients), and the information about the neighbouring subunits within the current macroblock (e.g., coded block flag). When a binary string is long, it is divided into parts conveyed in successive output words to the CABAC path. A

relevant part is selected using the barrel shifter driven by the register which identifies the number of released bits (invalid). In practice, some particular values are allowed, such as 0, 12, and multiplications of 7.
The second path, which supports VLC binarization schemas, concatenates code strings to form a codestream. The concatenation is performed in the VLC buffer and code strings are appended in successive clock cycles using a barrel shifter. Particularly, the shifter is driven by the number of valid bits kept in a separate register. It is increased by the length of a code string and decreased by the number of bits (eight-byte units) forwarded to the next stage.
The last fourth stage combines codestreams produced by the binarization and CABAC paths and encapsulates them into Network Abstraction Layer units. Note that data are accepted only from one path at a time depending on the selected mode and the processing state. The encapsulation amounts to adding one-byte header and the start code byte sequence at the beginning of each slice and sequence/picture headers. Additionally, an emulation prevention three byte (0x03) has to be inserted into the codestream when there is a forbidden byte sequence encountered. To facilitate the insertion process, previous pipeline stages (including CABAC path) are halted for one clock cycle. A dedicated subcircuit is responsible for the detection of the forbidden byte sequence. The subcircuit searches for 22 zero-valued bits starting from byte-aligned positions. All the processes are controlled by a dedicated FSM.

3.7.2 Arithmetic coding

The CABAC keeps up to 1024 probability models to increase the coding efficiency. Each type of syntax elements corresponds to a set of probability models pointed by different context labels. Each model is a Finite State Machine (FSM) that consists of the value of the more probable symbol (MPS) and the probability of the less probable symbol (LPS). The two variables are initialized based on the quantization parameter Qp with reference to the initialization set and the frame type. The FSMs are updated according to pre-defined adaptation rules. Context labels are computed as a sum of an offset ordered to a syntax element and an increment. Some increments are generated by referring to two adjacent macroblocks (16x16) or blocks (8x8 or 4x4) located on the left and the top of the current one. For other kinds of context labels, increments are formed on the basis of the previous bin value and the position in the binary string.
The main process in the CABAC is the recursive subdivision of a probability interval. In order to subdivide a probability interval length (range) into two subranges, probability estimates are determined on the basis of the probability model. The length of the first subinterval (LPS) is equal to the probability estimate, whereas that of the second one (MPS) is obtained by subtraction of the estimate from the current interval length. Depending on LPS/MPS coding, one of these subintervals is selected as a new interval length and renormalized to have the non-zero bit in the MSB position. While coding LPS, the subtraction outcome is added to the interval base (low). Successive renormalization shifts for the interval length trigger analogous modifications of the interval base. Bits released from MSB positions of the interval base drive the codestream formation process.
As some contexts are generated with reference to two adjacent macroblocks located to the left and on the top of the current one, the information relevant to form future contexts is

stored in registers and a double-port RAM memory, respectively. Access to the memory is performed on the macroblock basis. In the memory, 29 bits are required.
The architecture of the context formatter embeds one processing stage with an additional output stage as shown in Fig 15.a. Input data are produced by the binarization block and stored in the FIFO buffer. Loading of these data into registers is controlled by the FSM. Transactions of the FSM are driven by the counter (COUNT) and values of bits in the binarized representation. For each binarized syntax element, the counter determines the position of the bit for which context is generated. In fact, the position indicates the number of bits that have already been processed. On the basis of the state of the FSM, the context offset corresponding to a given syntax element is generated. Several offset-increment pairs are generated and stored in a small buffer. The adjustment of the context generation ratio is achieved by reading two pairs from the buffer. Having processed a syntax element, the input registers (CUR REG) are reloaded by the data for the following syntax element. If the information in the left-neighbouring registers (LEFT REG) is no longer referenced, the registers are successively rewritten by states of relevant registers for the current macroblock. This information is also stored in the context memory when all data for the current macroblock are released.
The block diagram of the CABAC initialisation unit is depicted in Fig. 15.b. The unit sets states of the CABAC probability model prior to submitting context-symbol pairs form the context formatter. To perform this task, one pair consisting of an index and a binary value of MPS is generated in each clock cycle. Although the initialisation procedure stops the main coding routine of the CABAC, associated time intervals have a small impact on the throughput. The initialisation unit applies three pipeline stages. The first stage generates the address to the 4Kx16-bit ROM memory used to keep initialisation parameters for four sets of parameters (one for INTRA and three for INTER) for High Profile (460 contexts). The second stage computes the internal variable denoted as *preState* on the basis of the quantization parameter Qp and parameters read from the memory. The computation is accomplished with the use of the multiplication and addition units. Apart from this, the subtraction of an offset value from the address taken from the previous stage provides the context label. The third stage maps the *preState* variable onto a MPS value and an index.

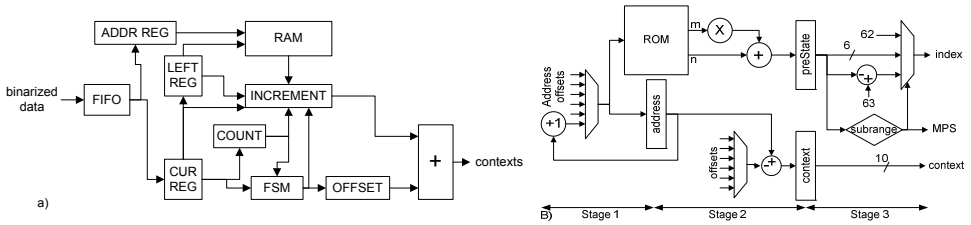

Fig. 15. Architectures of the context formatter (a) and the initialization unit (b)

The architecture of the arithmetic coder core with the enhanced bypass mode applies 9 pipeline stages (see Fig 16). This allows the minimization of critical paths and the adaptation to timing constraints resulting from reading the probability state memory. The first delay stage for input data is introduced to adjust input data to those read data from the probability state memory (addressed by the context label). As a consequence, the second

stage receives simultaneously values of contexts, symbols, indices, and the most probable symbols. On the basis of these variables, the circuit calculates a new indices and new values of the most probable symbols and stores them into the memory. Moreover, there are control signals to indicate either LPS or MPS coding. The memory operates at the doubled frequency to overcome problem of the simultaneous access to two entries corresponding to contexts ordered to symbols submitted in the same clock cycle of the main clock. If any same context labels are submitted in consecutive clock cycles, the first stage takes actual indices and MPS values from the following stage to keep the data consistency.

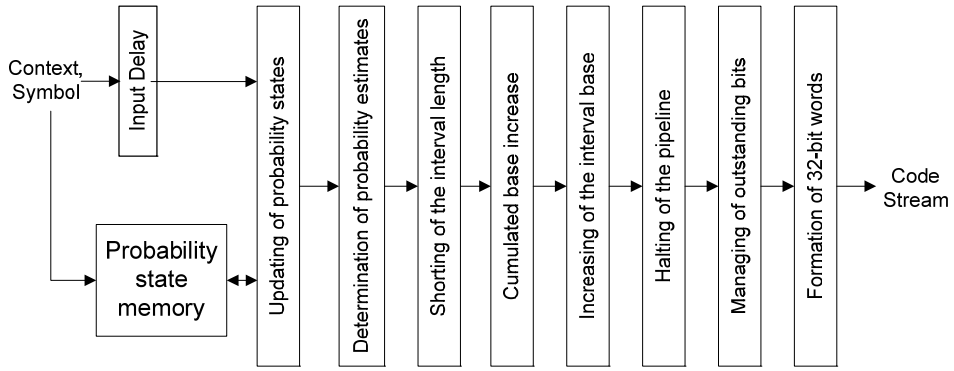

Fig. 16. Arithmetic Coding Pipeline

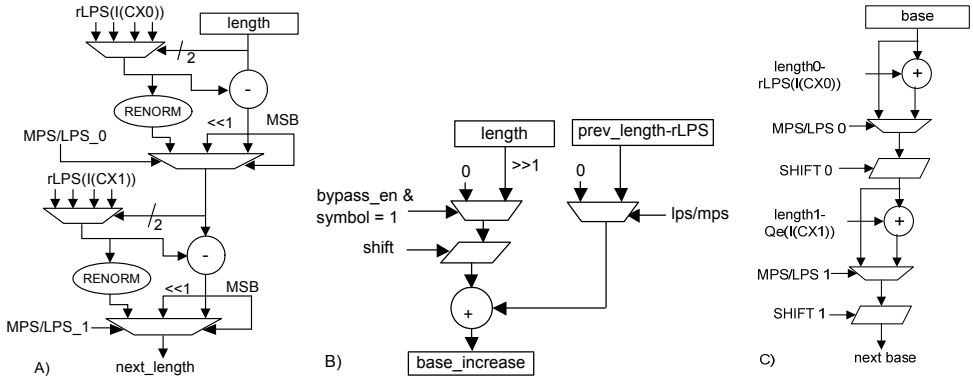

Fig. 17. Arithmetic Coding Stages 4th-6th

The LPS/MPS signal, along with the old index value, is forwarded to the third stage, which calculates probability estimates rLPS using four LUTs. The next stage reduces the interval length as shown in Fig. 17.a. The fifth stage computes the cumulated variables corresponding to the regular and bypass-mode symbols as shown in Fig. 17.b. They are used to increase the base register at the sixth stage (see Fig. 17.c). Bits released from this register are formed into codestream at the eight stage. Here, the outstanding bit counter collects series of ones and looks for a zero-valued bit or a carry to produce a part of the codestream.

It may occur that the number of bits to release is greater than the buffer size in the following stage. Such an event implicates the insertion of wait states, which stop all preceding pipeline stages, and the context-formation unit. A hold signal is driven directly by a register to optimize the clock rate. This involves a one-clock-cycle delay, which in turn imposes the use of an additional seventh stage to prevent loses of data between stopped and unstopped registers. The final tenth stage collects codestream into 32-bit words and releases them outside of the CABAC block.

3.8 Deblocking

The deblocking filter is applied to minimize artefacts on block/macroblock boundaries along both horizontal and vertical edges. The filtering is a two-phase non-linear operation that affects samples adjacent to boundaries and sometimes also their direct neighbours. Both phases are similar. In the first phase, the horizontal filter operates on vertical edges, whereas the vertical filter operates on horizontal edges in the second phase. The deblocking-filter data path is shown in Fig. 18. The module accepts one sample per clock cycles and the same throughput is at the output. Samples are carried by the pipeline registers. When a block edge samples are in q0 and p0 registers, the filter is activated (writing samples form the filter logic to registers). Since macroblocks are coded in the raster order, it sis necessary to incorporate a dedicated memory to buffer four picture lines (line of MB) for the filtering horizontal edges between macroblocks. One macroblock memory (MB1) is used to transpose the horizontally-filtered samples before the vertical filter. Another one (MB2) keeps left neighbouring samples form the previous macroblock.

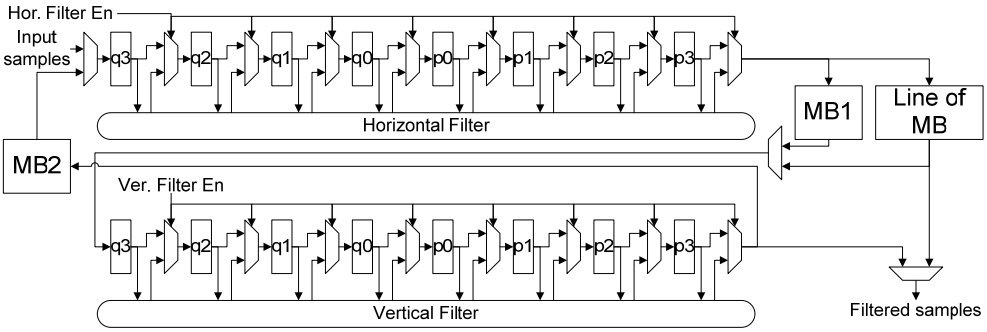

Fig. 18. Dataflow in the deblocking filter

There are four filter strengths, and the selection depends on two variables written into the codestream, the quantization parameters (alpha and beta), and edge type (macroblock or block). Horizontal and vertical filter logic embeds all the functionality that modifies samples based on the filter strength value. In particular, the non-linear filter logic analyzes input samples according to predefined formulas and compares the result with thresholds determined by the filter strength. If the threshold is exceeded, the filter is activated.

4. Implementation results

There are many complete video coding solutions developed by the scientific teams and commercial companies. The performance and resource cost is summarized in Section 2.3 for some of H.264/AVC encoders. This Section provides the implementation results of the developed architecture for key modules and compares them with other works.
Table 3 summarizes the resource consumption for modules described in Section 3. Note that the full encoder architecture needs more resources for the control and additional buffering between some modules. Moreover, the real hardware implementation requires some communication interfaces, i.e., the external memory controller, the codestream port, and the configuration port. Maximal clock rates obtained for the architecture are equal 100 MHz and 250 MHz for Aria and TSMC technologies, respectively.
Compared to other designs (see Section 2.3.), the developed architecture needs more on-chip memoires. The higher memory consumption results from the buffers incorporated to support the mode selection based on the rate-distortion analysis. This feature makes the architecture suitable for FPGA devices equipped with a significant amount of on-chip memories. Compared other designs, the logic consumption is relatively low when taking into account the encoder capability. Particularly, it can support High Profile options and HDTV at 200 MHz. Moreover, the advanced mode selection based on the rate-distortion criteria allows a better compression ratio for a given bit rate.

Module	Aria II [ALUT]	TSMC 0.13 µm [gate]	Memory [Kbit]
INTER PRED.	18756	140413	1300
INTRA PRED.	4599	23197	64
DCT	5279	37178	0
IDCT	5468	65869	0
QUANT	32xDSP+5236	78038	0
DEQUANT	32xDSP+2221	35421	0
RECONSTR.	3175	23420	0
DEBLOCKING	2395	17910	26
MODE SEL.	1xDSP+10646	39333	148
ENTROPY	66xDSP+6682	33206	105
ENCODER	72419	673256	2250

Table 3. Resource consumption for main modules of the hardware video encoder

5. Conclusion

The complexity of the state-of the-art video compression is high. The real-time performance requires the use of most advanced IC technologies to support high-definition

resolutions. Additionally, the need for buffering at different processing steps requires on-chip and external memories. The improvement in the compression efficiency requires more resources to tests many prediction modes and perform the rate-distortion analysis. The architecture described in the chapter is still developed. Particularly, it includes more advanced methods for the mode selections (alternative distortion measures, different quantization parameters, adaptive quantization), multi-view coding, and the robust rate control.

6. Acknowledgment

The work presented was developed within the research project LIDER/05/8/L-2/10/NCBiR/2011 founded by the Notional Centre for Research and Development, Warsaw, Poland.

7. References

Chen Y.-H., Chen T.-C., Tsai C.-Y., Tsai S.-F., & Chen L.-G.; Algorithm and Architecture Design of Power-Oriented H.264/AVC Baseline Profile Encoder for Portable Devices, *IEEE Transactions on Circuits and Systems for Video Technology*, vol.19, no.8, pp.1118-1128, ISSN 1051-8215, Aug. 2009

Jakubowski, M. & Pastuszak, G.; (2008). Data Reuse in Two-Level Hierarchical Motion Estimation for High Resolution Video Coding, *Proceedings of SIGMAP 2010 International Conference on Signal Processing and Multimedia Applications*, pp. 159-162, Athens, Greece, July 26-28, 2010

Lin Y.-K., Ku C.-W., Li D.-W., & Chang, T.-S.; (2009). A 140-MHz 94 K Gates HD1080p 30-Frames/s Intra-Only Profile H.264 Encoder. *IEEE Transactions on Circuits and Systems for Video Technology,* Vol.19, No.3, (March 2009), pp. 432-436, ISSN 1051-8215

Lin Y.-L. S, Kao C.-Y., Kuo H.-C., & Chen J.-W., VLSI Design for Video Coding: H.264/AVC Encoding from Standard Specification to Chip, *Springer*, 2010, ISBN 978-1-4419-0958-9

Liu Z.; Song Y., Shao M., Li S., Li L., Ishiwata S., Nakagawa M., Goto S. & Ikenaga, T.; HDTV1080p H.264/AVC Encoder Chip Design and Performance Analysis, *IEEE Journal of Solid-State Circuits*, vol.44, no.2, pp.594-608, Feb. 2009

Pastuszak, G.; (2008). Transforms and Quantization in the High-Throughput H.264/AVC Encoder Based on Advanced Mode Selection, *Proceedings of ISVLSI 2008 IEEE Annual Symposium on VLSI*, pp. 14-17, Montpellier, France, April 7-9, 2008

Pastuszak, G. (2008). A High Performance Architecture of the Double-Mode Binary Coder for H.264.AVC. *IEEE Transactions on Circuits and Systems for Video Technology,* Vol.18, No.7, (July 2008), pp. 949-960, ISSN 1051-8215

Roszkowski, M.; Abramowski, A.; Wieczorek, M. & Pastuszak, G. (2010). Architecture design of the hardware H.264/AVC video decoder. *Journal of Electronics and Telecommunications,* Vol.55, No.3, (3/2010), pp. 291-300, ISSN 0867-6747

Roszkowski, M. & Pastuszak G.; (2010). Intra Prediction Hardware Module for High-Profile H.264/AVC Encoder, *Signal Processing - Algorithms, Architectures, Arrangements, and Applications* (SPA 2010), Poznań, Poland, 23-25 September 2010.

Y. W. Huang, B. Y. Hsieh, T. C. Chen, and L. G. Chen, "Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 3, pp. 378–401, Mar. 2005.

Cutting Edge Research in New Technologies

Edited by Prof. Constantin Volosencu

ISBN 978-953-51-0463-6
Hard cover, 346 pages
Publisher InTech
Published online 05, April, 2012
Published in print edition April, 2012

The book "Cutting Edge Research in New Technologies" presents the contributions of some researchers in modern fields of technology, serving as a valuable tool for scientists, researchers, graduate students and professionals. The focus is on several aspects of designing and manufacturing, examining complex technical products and some aspects of the development and use of industrial and service automation. The book covered some topics as it follows: manufacturing, machining, textile industry, CAD/CAM/CAE systems, electronic circuits, control and automation, electric drives, artificial intelligence, fuzzy logic, vision systems, neural networks, intelligent systems, wireless sensor networks, environmental technology, logistic services, transportation, intelligent security, multimedia, modeling, simulation, video techniques, water plant technology, globalization and technology. This collection of articles offers information which responds to the general goal of technology - how to develop manufacturing systems, methods, algorithms, how to use devices, equipments, machines or tools in order to increase the quality of the products, the human comfort or security.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Grzegorz Pastuszak (2012). Video Compression from the Hardware Perspective, Cutting Edge Research in New Technologies, Prof. Constantin Volosencu (Ed.), ISBN: 978-953-51-0463-6, InTech, Available from: http://www.intechopen.com/books/cutting-edge-research-in-new-technologies/video-compression-from-the-hardware-perespective

INTECH
open science | open minds

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