12

An FPGA Implementation of HW/SW Codesign Architecture for H.263 Video Coding

A. Ben Atitallah[1,2], P. Kadionik[2], F. Ghozzi[1], P.Nouel[2],
N. Masmoudi[1] and H. Levi[2]
[1]*Laboratory of Electronics and Information Technology National Engineers School of Sfax (E.N.I.S.), BP W 3038 Sfax*
[2]*IXL laboratory –ENSEIRB - University Bordeaux 1 - CNRS UMR 5818, 351 Cours de la Libération, 33 405 Talence Cedex,*
[1]*Tunisia*
[2]*France*

1. Introduction

Video is fundamental component of a wide spectrum of the multimedia embedded systems. The great interest for digital as opposed to analog video is because it is easier to transmit access, store and manipulate visual information in a digital format. The key obstacle to using digital video is the enormous amount of data required to represent video in digital format. Compression of the digital video, therefore, is an inevitable solution to overcome this obstacle. Consequently, academia and industry have worked toward developing video compression algorithms [1]-[3], which like ITU-T H.261, H.263, ISO/IEC MPEG-1, MPEG-2 and MPEG-4 emerged with a view to reduce the data rate to a manageable level by taking advantage of the redundancies present both spatial and temporal domains of the digital video.

The ITU-T H.263 standard is an important standard for low bit rate video coding, enabling compression of video sequences to a bit rate below 64 kbps [4]. H.263 coding algorithm can be found in different application such as videoconferencing, videophone and video emailing. Due to the different profiles and levels of the H.263 standard, every application can have specific ratio between performance and quality. Since modern digital video communication applications increase both aspects of the H.263 compression, it is therefore necessary to achieve best performance in terms of real-time operation.

The H.263 standard includes several blocks such as Motion Estimation (ME), Discrete Cosine Transform (DCT), quantization (Q) and variable length coding (VLC). It was shown that some of these parts can be optimized with parallel structures and efficiently implemented in hardware/software (HW/SW) partitioned system. However, there exists a trade-off between hardware and software implementation. Various factors such as flexibility, development cost, power consumption and processing speed requirement should be taken into account. Hardware implementation is generally better than software implementation in processing speed and power consumption. In contrast, software can give a more flexible design solution and also be more suitable for various video applications [5].

Recently, several H.263 encoders have been developed either as software based applications [6]-[7] or hardware based VLSI custom chips [8].
In [6], the H.263 implementation based on general purpose microprocessors, including PCs or workstations. All the efforts are focused on the optimization of the code that implements the encoder for the target microprocessor. In [7], an architecture based on a Digital Signal Processor (DSP) is described to implement a real-time H.263 encoder using fast algorithms to reduce the encoder computational complexity. Architecture based on a dedicated sequencer and a specialized processor is detailed in [8]. It is implemented on Xilinx FPGA and carrying out the basic core of H.263 without motion estimation. The depicted architectures lack flexibility because of their dedicated controller.
In literature, we haven't found any description of combined HW/SW implementation of the H.263 encoder on a single chip. The reason is probably the lack of technology that provides efficient HW/SW implementation. With the recent advantages in technology from leading manufacturers of the programmable devices, such as Xilinx [9] and Altera [10], the proposed approach gains importance. In order to take advantages of both software and hardware implementation, each functional module of the H.263 video encoder is studied to determine a proper way for HW/SW partitioning. Based on this study, DCT and inverse DCT (IDCT) algorithm are implemented with fast parallel architectures directly in hardware. Also, the quantization and inverse quantization (IQ) are implemented in hardware using NIOS II custom instruction logic. These parts are described in VHDL (*VHSIC Hardware Description language*) language and implemented with the NIOS II softcore processor in a single Stratix II EP2S60 FPGA (*Field Programmable Gate Array*) device and the remaining parts are performed in software on NIOS II softcore processor and using μClinux , an embedded Linux flavour, as operating system. This partitioning has been chosen in order to achieve better timing results.
This paper is organized as follows: section 2 describes the baseline H.263 video encoder. Section 3 presents the HW/SW codesign platform. Timing optimization of the H.263 encoder using the HW/SW codesign is described in section 4. The design environment and FPFA implementation of the encoder is presented in section 5. The experiment results are shown in section 6. Finally, section 7 concludes the paper.

2. Baseline H.263 video coding

The coding structure of H.263 is based on H.261 [11]. In these standards, motion estimation and compensated are used to reduce temporal redundancies. DCT based algorithms are then used for encoding the motion compensated prediction difference frames. The quantized DCT coefficients, motion vector and side information are entropy coded using variable length codes. In this section, one describes first the picture formats used by H.263 encoders and the organization of pictures into smaller structures. It then reviews the general coding principles used by this encoder and describes their different blocks.

A. Picture format and organization

H.263 supports five standardized picture formats: CIF (Common Intermediate Format), 4CIF, 16CIF, QCIF (quarte-CIF) and sub-QCIF. Custom picture formats can also be negotiated by the encoder. However only the QCIF and sub-QCIF are mandatory for an H.263 decoder and the encoder only needs to support one of them.

The luminance component of the picture is sampled at these resolutions, while the chrominance components, Cb and Cr, are downsampled by two in both the horizontal and vertical directions. The picture structure is shown in Fig.1 for the QCIF resolution. Each picture in the input video sequence is divided into macroblocks (MB), consisting of four luminance blocks of 8 pixels x 8 lines followed by one Cb block and one Cr block, each consisting of 8 pixels x 8 lines. A group of blocks (GOB) is defined as an integer number of MB rows, a number that is dependent on picture resolution. For example, a GOB consists of a single MB row at QCIF resolution.

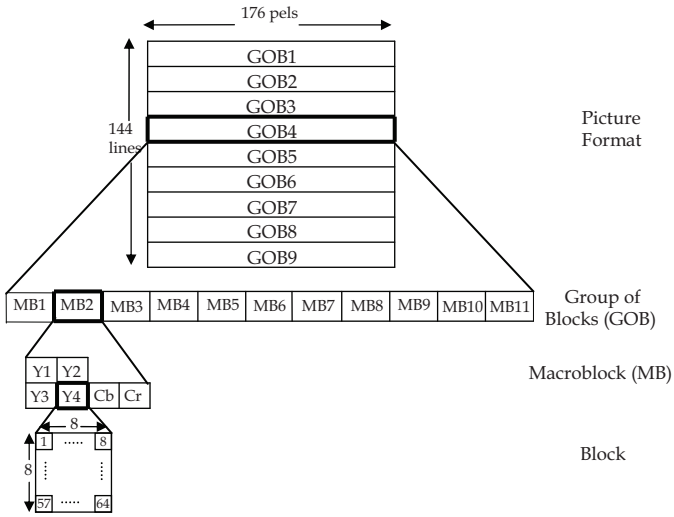


Fig. 1. H.263 picture structure for QCIF resolution

B. Overview of the H.263 video coding standard

The block diagram of an H.263 baseline video encoder is shown in Fig.2. The encoder operation is based on hybrid differential/transform coding, and is a combination of lossy and lossless coding. There are two fundamental modes which are jointly used for maximum compression efficiency: the intra and inter modes. Different types of frames correspond to these modes.

In the intra mode, the contents of a video frame are first processed by a DCT. The resulting coefficients are quantized with a chosen quantizer step size, thus leading to a loss of information. The quantized DCT coefficients are entropy coded using VLC, scanned across the picture (often using a zig-zag strategy), and delivered to an encoder buffer. The intra mode produces intra frames (I-frames). This kind of frame is needed for the decoder to have a reference for prediction. However, I-frames use a large amount of bits, so that they should be used sparingly in low bit rate applications. In the inter mode, the same operations are applied to the motion-predicted difference between the current frame and the previous (or earlier) frame, instead of the frame itself. To this end a motion estimation algorithm is applied to the input frame, and the extracted motion information (in the form of motion vectors, MV) is used in predicting the following frames, through a motion-compensation bloc. In order to avoid a drift between the encoder and decoder due to motion prediction,

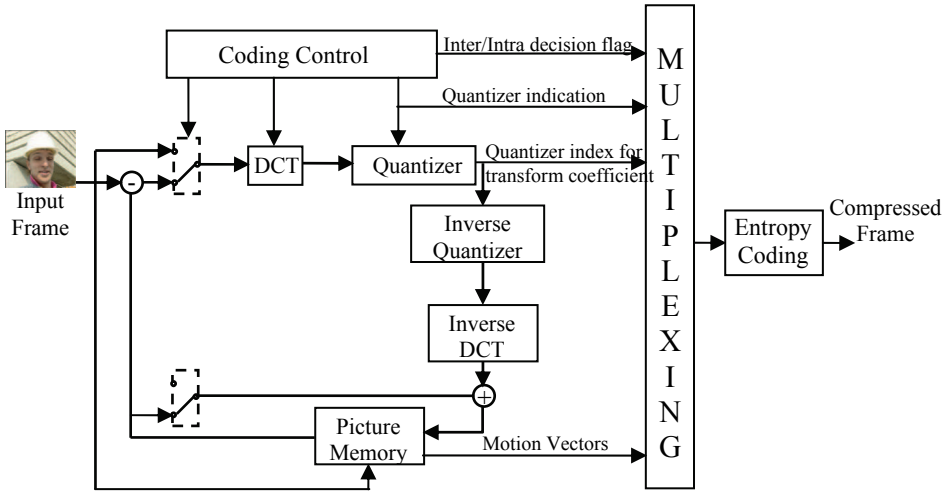


Fig. 2. Baseline H.263 video encoder block diagram

the motion compensation bloc needs to use a locally reconstructed version of the compressed frame being sent: this explains the presence of an inverse quantizer and an inverse discrete cosine transform in the feedback loop. The MV is differentially coded in order to realize bit rate savings. The inter mode produces prediction frames (P-frames) which can be predicted from I-frames or others P-frames. These in general use considerably less bits than I-frames, and are responsible for the large compression gain.

1) Motion estimation and compensation: It is often the case that video frames that are close in time are also similar. Therefore, when coding a video frame, it would be judicious to make as much use as possible of the information presented in a previously coded frame. One approach to achieve this goal is to simply consider the difference between the current frame and a previous reference frame, as shown in Fig. 3, and code the difference or residual. When the two frames are very similar, the difference will be much more efficient to code than coding the original frame. In this case, the previous frame is used as an estimate of the current frame.

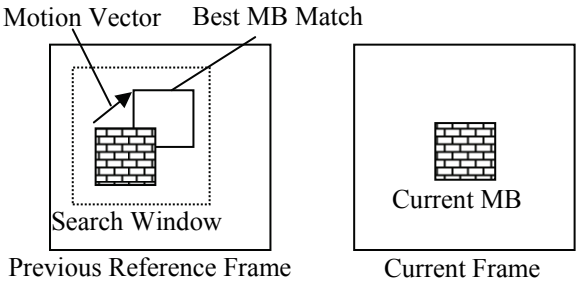


Fig. 3. Block-matching algorithm

A more sophisticated approach to increase coding efficiency is to work at the macroblock level in the current frame, instead of processing the whole frame all at once as described

above. The process is called motion compensated prediction, and is based on the assumption that most of the motion that the macroblocks undergo between frames is a translational motion. This approach attempts to find, for each 16x16 luminance block of a macroblock in the current frame, the best matching block in the previous frame. A search window is usually defined and bounds the area within which the encoder can perform the search for the best matching block. The motion of a macroblock is represented by a motion vector that has two components; the first indicating horizontal displacement, and the second indicating vertical displacement. Different criteria could be used to measure the closeness of two blocks [12]. The most popular measure is the Sum of Absolute Differences (SAD) defined by

$$SAD = \sum_{i=0}^{15}\sum_{j=0}^{15}\left|Y_{k,l}(i,j) - Y_{k-u,l-v}(i,j)\right| \tag{1}$$

Where $Y_{k,l}(i,j)$ represents the $(i,j)th$ pixel of a 16 x 16 MB from the current picture at the spatial location (i,j) and $Y_{k-u,l-v}(i,j)$ represents the $(i,j)th$ pixel of a candidate MB from a reference picture at the spatial location (k,l) displaced by the vector (i,j). To find the macroblock producing the minimum mismatch error, we need to compute SAD at several locations within a search window. This approach is called full search or exhaustive search, and is usually computationally expensive, but on the other hand yields good matching results.

2) DCT Transform: The basic computation element in a DCT-based system is the transformation of an NxN image block from the spatial domain to the DCT domain. For the video compression standards, N is usually 8. The 8 x 8 DCT is simple, efficient and well suited for hardware and software implementations. The 8 x 8 DCT is used to decorrelate the 8 x 8 blocks of original pixels or motion compensated difference pixels and to compact their energy into few coefficient as possible. The mathematical formulation for the (two-dimensional) 2-D DCT is shown in equation (2) [13].

$$\begin{aligned} y_{k,l} &= \frac{c(k)c(l)}{4}\sum_{i=0}^{7}\sum_{j=0}^{7} x_{i,j}\cos\left(\frac{(2i+1)k\pi}{16}\right)\cos\left(\frac{(2j+1)l\pi}{16}\right) \\ with \quad c(k),c(l) &= \frac{1}{\sqrt{2}}(k,l=0)...otherwise \quad 1 \end{aligned} \tag{2}$$

The 2-D DCT in (2) transforms an 8 x 8 block of pictures samples $x_{i,j}$ into spatial frequency components $y_{k,l}$ for $0 \le k,\ j \le l$. The 2-D IDCT in (3) performs the inverse transform for $0 \le i, j \le 7$.

$$x_{i,j} = \frac{1}{4}\sum_{k=0}^{7}\sum_{l=0}^{7} y_{k,l}c(k)c(l)\cos\left(\frac{(2k+1)i\pi}{16}\right)\cos\left(\frac{(2l+1)j\pi}{16}\right) \tag{3}$$

Although exact reconstruction can be theoretically achieved, it is often not possible using finite-precision arithmetic. While forward DCT errors can be tolerated, IDCT errors must meet the H.263 standard if compliance is to be achieved.

3) Quantization: The quantization is a significant source of compression in the encoder bit stream. Quantization takes advantage of the low sensitivity of the eye to reconstruction

errors related to high spatial frequencies as opposed to those related to low frequencies [14]. Quick high frequency changes can often not be seen, and may be discarded. Slow linear changes in intensity or color are important to the eye. Therefore, the basic idea of the quantization is to eliminate as many of the nonzero DCT coefficients corresponding to high frequency components.

Every element in the DCT output matrix is quantized using a corresponding quantization value in a quantization matrix. The quantizers consist of equally spaced reconstruction levels with a dead zone centered at zero. In baseline H.263, quantization is performed using the same step size within a macroblock by working with a uniform quantization matrix. Except for the first coefficient of INTRA blocks is nominally the transform DC value uniformly quantized with a step size of eight, even quantization levels in the range from 2 to 62 are allowed. The quantized coefficients are then rounded to the nearest integer value. The net effect of the quantization is usually a reduced variance between the original DCT coefficients as compared to the variance between the original DCT coefficients. Another important effect is a reduction in the number of nonzero coefficients.

4) Entropy coding: Entropy coding is performed by means of VLC, and is used to efficiently represent the estimated motion vectors and the quantized DCT coefficients. Motion vectors are first predicted by setting their component values to median values of those of neighboring motion vectors already transmitted: the motion vectors of the macroblocks to the left, above, and above right of the current macroblock. The difference motion vectors are then VLC coded.

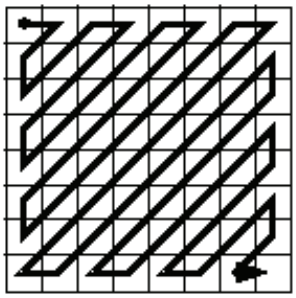


Fig. 4. Zig-zag positioning of quantized transform coefficients

As for the quantized DCT coefficients, they are first converted into a one-dimensional array for entropy coding by an ordered zigzag scanning operation. The resulting array contains a number of nonzero entries and probably many zero entries. This rearrangement places the DC coefficient first in the array, and the remaining AC coefficients are ordered from low to high frequency. This scan pattern is illustrated in Fig. 4. The rearrangement array is coded using three parameters (LAST, RUN, LEVEL). The symbol RUN is defined as the distance between two nonzero coefficients in the array (i.e., the number of zeros in a segment). The symbol LEVEL is the nonzero value immediately following a sequence of zeros. The symbol LAST, when set to 1, is used to indicate the last segment in the array. This coding method produces a compact representation of the 8x8 DCT coefficients, as a large number of the coefficients are normally quantized to zero and the reordering results (ideally) in the grouping of long runs of consecutive zero values. Other information such as prediction types and quantizer indication is also entropy coded by means of VLC.

3. The HW/SW codesign platform

A. FPGA platform

Field Programmable Devices are becoming increasingly popular for implementation of digital circuits. The case of FPGA is the most spectacular and is due to several advantages, such as their fast manufacturing turnaround time, low start-up costs and particularly ease of design. With increasing device densities, audacious challenges become feasible and the integration of embedded SoPC (System on Programmable Chip) systems is significantly improved [15].

Furthermore, reconfigurable systems on a chip became a reality with softcore processor, which are a microprocessor fully described in software, usually in a VHDL, and capable to be synthesized in programmable hardware, such as FPGA. Softcore processors can be easily customized to the needs of a specific target application (e.g. multimedia embedded systems). The two major FPGA manufacturers provide commercial softcore processors. Xilinx offers its MicroBlaze processor [16], while Altera has Nios and Nios II processors [17]. The benefit of a softcore processor is to add a micro-programmed logic that introduces more flexibility. A HW/SW codesign approach is then possible and a particular functionality can be developed in software for flexibility and upgrading completed with hardware IP blocks (Intellectual Property) for cost reduction and performances.

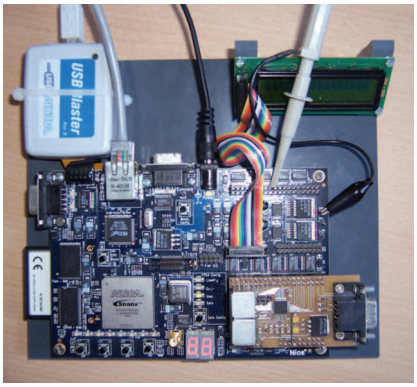


Fig. 5. Stratix II Development Board

B. The NIOS II development board - the HW/SW platform

For SW implementation of image and video algorithms, the use of a microprocessor is required. The use of additional HW for optimization contributes to the overall performance of the algorithm. For the highest degree of HW/SW integration, customization and configurability, a softcore processor was used.

For the main processing stage, the Altera NIOS II development board was chosen (Fig. 5) [18]. The core of the board is the Altera Stratix II EP2S60F672C3 FPGA. Several peripheral devices and connectors (UART, LCD, VGA, Ethernet etc) serve as interfaces between the Stratix II FPGA and the external environment. 8MByte FLASH, 16MByte SRAM and 1MByte SRAM allow implementation of complex FPGA video applications. For the video embedded systems, we are using flash memory, SRAM, SDRAM, UART, timer, Ethernet and Camera for frame acquisition.

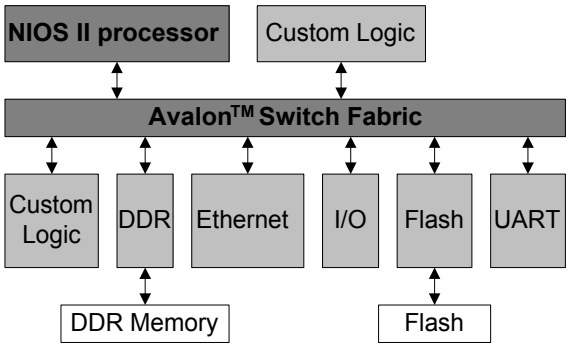


Fig. 6. NIOS II embedded system

Altera introduces the SOPC builder tool [19], for the quick creation and easy evaluation of embedded systems. The integration off-the-shelf intellectual property (IP) as well as reusable custom components is realized in a friendly way, reducing the required time to set up a SoPC and enabling to construct and designs in hours instead of weeks. Fig. 6 presents the Stratix II FPGA with some of the customizable peripherals and external memories, as an example of their applicability.

1) NIOS II CPU: The Altera NIOS II softcore processor (*FAST version*) is a 32-bits scalar RISC with Harvard architecture, 6 stages pipeline, 1-way direct-mapped 64KB data cache, 1-way direct-mapped 64KB instruction cache and can execute up to 150 MIPS [17].

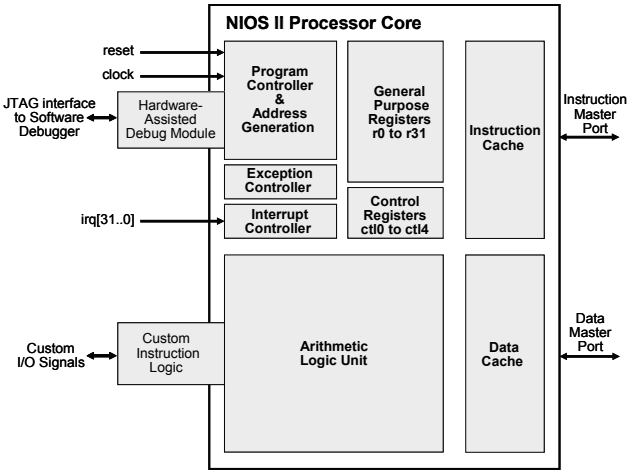


Fig. 7. NIOS II processor core block diagram

The main interest of this softcore processor is its extensibility and adaptability. Indeed, users can incorporate custom logic directly into the NIOS II Arithmetic Logic Unit (ALU) [20]. Furthermore, users can connect into the FPGA the on-chip processor and custom peripherals to a dedicated bus (Avalon Bus). Thus, users can define their instructions and processor peripherals to optimize the system for a specific application. Fig.7 show the block diagram of the NIOS II softcore processor core which defines the following user-visible

functional units: register file, arithmetic logic unit, interface to custom instruction logic, interrupt controller, instruction and data bus, instruction and data cache memories and JTAG debug module.

2) *NIOS II custom instruction logic:* With Nios II custom instructions [21], system designers are able to take full advantage of the flexibility of FPGA to meet system performance requirements. Custom instructions allow system designers to add up to 256 custom functionalities to the Nios II processor ALU. As shown in Fig.8, the custom instruction logic connects directly to the Nios II ALU (Arithmetic Logic Unit). There are different custom instruction architectures available to suit the application requirements. The architectures range from simple, single-cycle combinatorial architectures to extended variable-length, multi-cycle custom instruction architectures. The most common type of implementation is the single-cycle combinatorial architecture that allows for custom logic realizations with one or two inputs and one output operand.

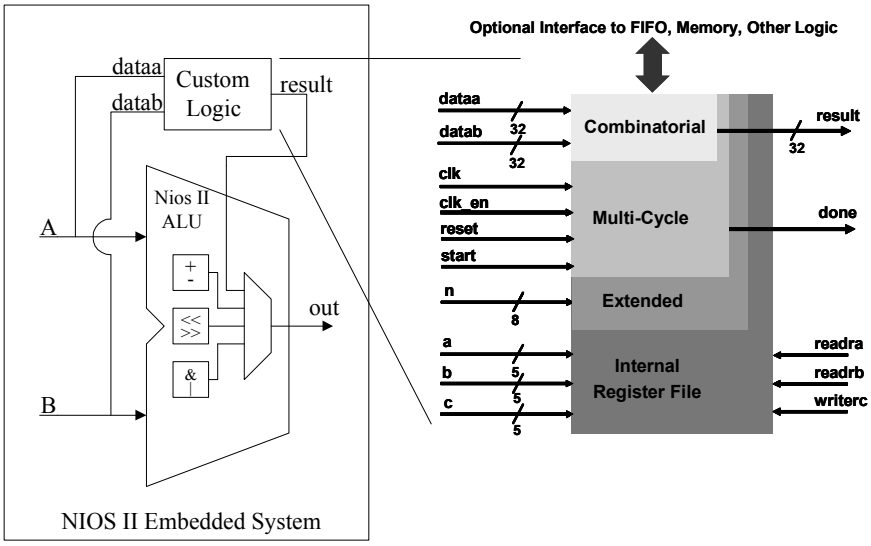


Fig. 8. Custom Instruction Logic Connects to the Nios II ALU

Using the custom instruction in an ANSI C program is straightforward. Two kinds of define macros are used as the instructions can have one or two input operands:

- #define INSTR1(X) __builtin_custom_ini(Code_INSTR1,X)
- #define INSTR2(X,Y) __builtin_custom_ini(Code_INSTR2,X,Y)

C) The HW/SW codesign process

The HW/SW codesign process for the implementation into the platform can be summarized in three main steps:

- Algorithm implementation in SW.
- Detecting critical software parts.
- HW/SW optimization of the algorithm.

The first step is implementing algorithm in SW. The ANSI C language and the assembly programming language are supported. Generally, the preferable choice is the

implementation of the SW code using ANSI C. In this way, instead of rewriting the code from scratch, the use of an already existing code for the algorithm shortens the design cycle. The portability of ANSI C allows also the code to be created and tested for functionality on other platforms.
Once the SW code has been tested for functionality and implemented into the target platform, the performance analysis has to be applied. In order to reach the required constraints, critical software parts has to be detected and optimized. To have precision on the time processing, a CPU timer can be used for the cycle-accurate time-frame estimation of a focused part of the SW code execution.
The final step is the SW code refinement and optimization of critical SW parts using HW description. The general idea is to implement parallel structures in HW for fastest data processing. The SW and HW parts are dependent and, regarding the interface between them, can be incorporated into the algorithm as separate HW component (access register) or custom instruction (the custom instruction is integrated directly into CPU as an additional instruction).
In the HW/SW codesign process, the designer iterates through the last two design steps until the desired performance is obtained.

D) Using embedded linux with codesign

The HW/SW codesign process uses different kinds of peripherals and memories. The basic idea is to use Linux in an embedded system context. Linux for embedded systems or embedded Linux gives us several benefits:

- Linux is ported to most of processors with or without Memory Management Unit (MMU). A Linux port is for example available for the NIOS II softcore.
- Most of classical peripherals are ported to Linux.
- A file system is available for data storage.
- A network connectivity based on Internet protocols is well suited for data recovering.
- Open source Linux projects may be used.

The embedded Linux environment is also a real advantage for the software development during the HW/SW codesign process.

4. Timing optimisation of the H.263 encoder

A) Timing optimisation

In order to optimize and achieve best performance in terms of real-time operation of the H.263 video encoder, we have used the HW/SW codesign process. At first, the algorithms were coded in ANSI C programming language on a PC platform. The tested SW code was then rebuilt and transferred into the Nios II system. The execution times have been measured with the high_res_timer that provides the number of processor clock cycles for the execution time. Afterwards, the SW critical parts were implemented in HW in VHDL language.
In our experiments of coding a general video clip in QCIF (Quarter Common Intermediate Format: Spatial resolution of 176x144 and temporal resolution 10 frames/s (fps)) format. The average frame rate achieved on a NIOS II system is only about 0.7 fps. For this reason, we investigated the resource distribution of the H.263 video encoder which uses full search motion estimation, search window size +/-7 and fixed quantization parameters QP=16. Fig.9 shows the distribution of the execution time of Miss America and Carphone sequences.

In this figure ME, DCT/IDCT and Q/IQ which utilize 23.1%-27.7%, 67.3-71.9% and 2%-2.2% of the total execution time respectively are the three primary computationally intensive components. Thus, main purpose is to improve these three components using HW/SW codesign

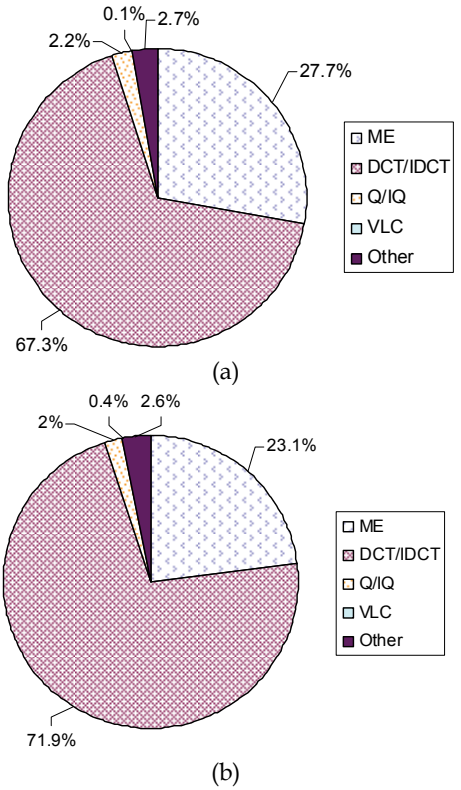


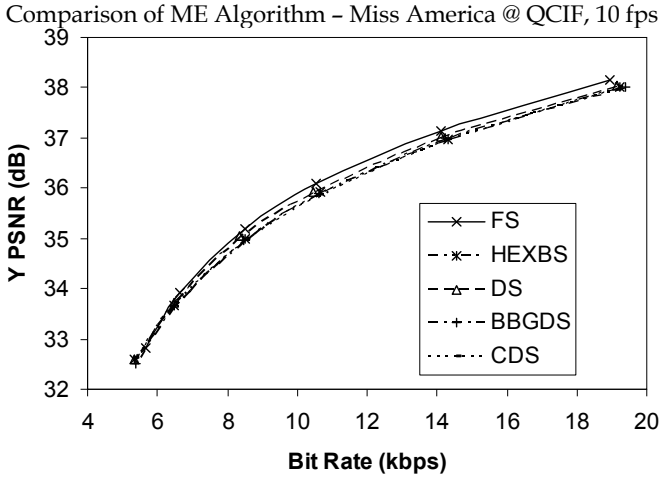
Fig. 9. Execution time distribution of (a) Miss America and (b) Carphone sequences at QCIF resolution

B) Hardware/Software partitioning

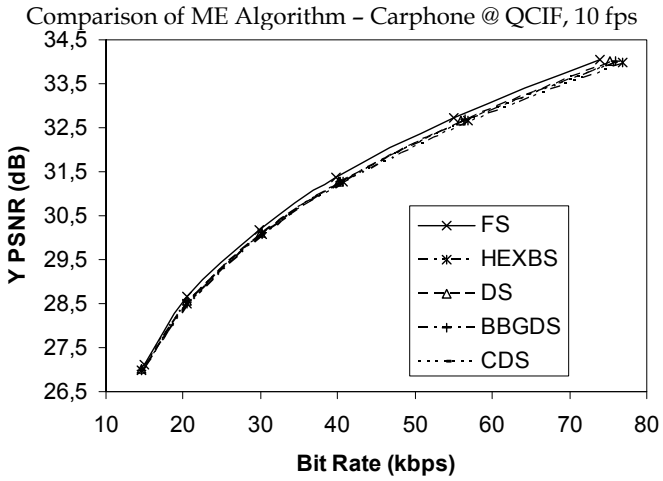
The main idea of our encoder is to exploit advantages of the parallel structures which can be efficiently implemented in hardware. Hardware implementation of 2-D DCT/IDCT and Q/IQ promise better results compared to software based algorithms. The key point of a parallel hardware structure is a reduced number of operation and ability to function parallel. However, there is still a good chance to reduce the complexity of the ME in software using fast motion estimation algorithms.

1) Optimization in Motion estimation: Motion estimation (ME) removes temporal redundancy between successive frames in digital video. The most popular technique for motion estimation is the block-matching algorithm [11]. Among the block-matching algorithms, the full search or exhaustive search algorithm examines all search points inside the search area. Therefore, the amount of its computation is proportion to the size of the search window.

Although it finds the best possible match, it requires a very large computational complexity (600 MOPS "million operations per second" for QCIF@10 Hz and +/-15 search window). Hence, many fast algorithms are proposed in literature such as the hexagon-based search [22], the diamond search [23]-[24], the block-based gradient descent search [25] and the cross-diamond search [26], which allow to reduce the computational complexity at the price of slightly performance loss. The basic principle of these fast algorithms is dividing the search process into a few sequential steps and choosing the next search direction according to the current search result.



(a)



(b)

Fig. 10. Comparison of motion estimation algorithms of: (a) Miss America and (b) Carphone at QCIF resolution and 10fps

In order to reduce the encoder computational complexity, we analyze the performance and speed of the different fast algorithms. The average peak signal-to-noise ratio (PSNR) is used as a distortion measure, and is given by

$$PSNR = 10\log\frac{1}{M}\sum_{n=1}^{M}\frac{255^2}{(o_n - r_n)} \tag{4}$$

Where M is the number of samples and o_n and r_n are the amplitudes of the original and reconstructed pictures, respectively. The average PSNR of all encoded pictures is here used as a measure of objective quality.

Fig.10 illustrates the rate-distortion performance of several popular block-matching algorithms namely full search (FS), hexagon-based search (HEXBS), diamond search (DS), block-based gradient descent search (BBGDS) and cross-diamond search (CDS) algorithms. Fig.11 presents the clock number necessary to perform these fast motion estimation algorithms (HEXBS, DS, BBGDS and CDS). For Miss America and Carphone sequences, we can conclude, using the HEXBS method, a 12.5 to 13 fold speed increase on motion estimation is achieved compared to the FS method whilst the PSNR degradation is marginal.

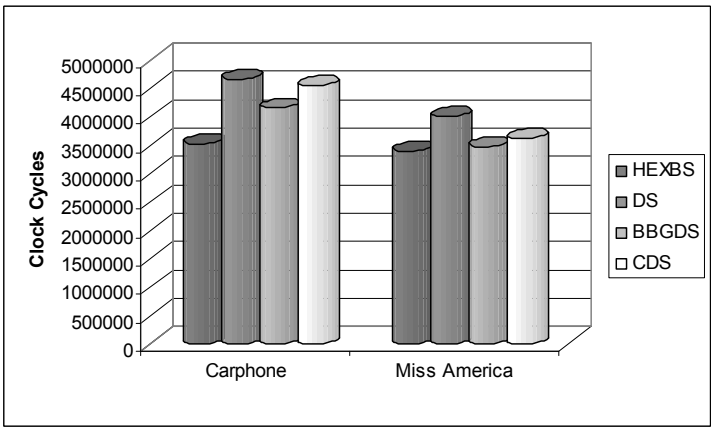


Fig. 11. Cycles required to perform the fast motion estimation algorithms

The HEXBS is the popular fast block-matching algorithms and it can reduce computational complexity. The HEXBS algorithm employs two search patterns as illustrated in Fig. 12. The first pattern, called *large hexagon search pattern* (LHSP), comprises seven checking points from which six points surround the center one to compose a hexagon shape. The second pattern consisting of five checking points forms a smaller hexagon shape, called *small hexagon search pattern* (SHSP).

In the searching procedure of the HEXBS algorithm, LHSP is repeatedly used until the step in which the minimum block distortion (MBD) occurs at the center point. The search pattern is then switched from LHSP to SHSP as reaching to the final search stage. Among the five checking points in SHSP, the position yielding the MBD provides the motion vector of the best matching block. Fig. 13 shows an example of the search path strategy leading to the motion vector (-3, -1), where 20 (7+3+3+4) search points are evaluated in 4 steps sequentially.

(a) Large hexagonal search pattern (LHSP) (b) Small hexagonal search pattern (SHSP)

Fig. 12. Two search patterns derieved are employed in the HS algorithm

The procedure of the HEXBS is described below:

Step 1. The initial LHSP is centered at the origin of the search window, and the 7 checking points of LHSP are tested. If the MBD point calculated is located at the center position, go to Step 3; otherwise, go to Step 2.

Step 2. The MBD point found in the previous search step is repositioned as the center point to form a new LHSP. If the new MBD point obtained is located at the center position, go to Step 3; otherwise, recursively repeat this step.

Step 3. Switch the search pattern from LHSP to SHSP. The MBD point found in this step is the final solution of the motion vector which points to the best matching-block.

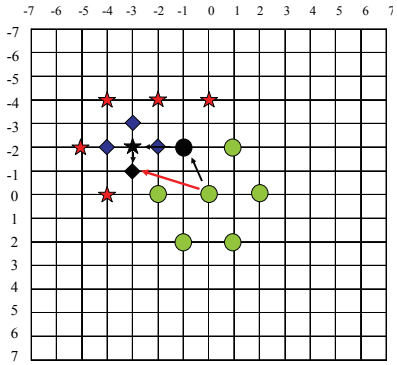


Fig. 13. Search path example which leads to the motion vector (-3, -1) in four search steps

2) Optimization in DCT and IDCT: Since the straightforward implementation of (2) and (3) are computationally expensive (with 4096 multiplications), many researches have been done to optimize the DCT/IDCT computational effort using the fast algorithms such as Lee [27], Chen [28] and Loeffler [29]. Most of the efforts have been devoted to reduce the number of operations, mainly multiplications and additions. In our DCT/IDCT hardware implementation, we use an 8-point one-dimensional (1-D) DCT/IDCT algorithm, proposed by van Eijdhoven and Sijstermans [30]. It was selected due the minimum required number of additions and multiplications (11 Multiplications and 29 additions). This algorithm is obtained by a slight modification of the original Loeffler algorithm [29], which provides one of the most computationally efficient 1-D DCT/IDCT calculation, as compared with other known algorithms [31]-[33]. The modified Loeffler algorithm for calculating 8-point 1-D DCT is illustrated in Fig.14.

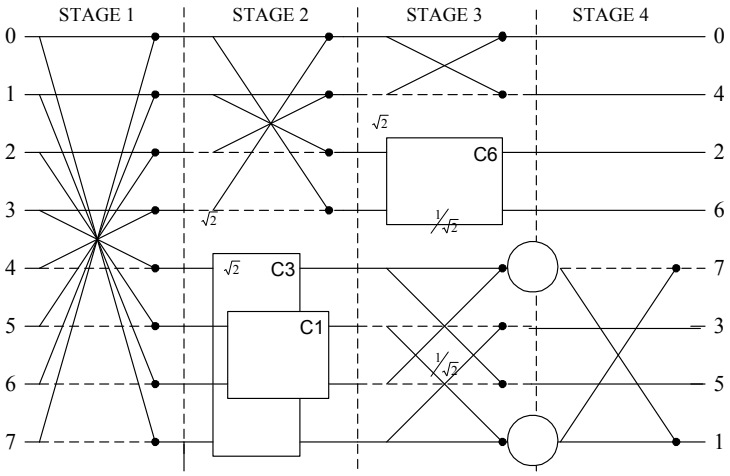
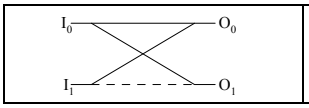


Fig. 14. The 8-point DCT modified Loeffler algorithm

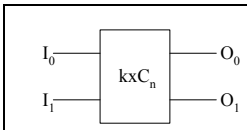
The stages of the algorithm numbered 1 to 4 are parts that have to be executed in serial mode due to the data dependency. However, computation within the first stage can be parallelized. In stage 2, the algorithm splits in two parts: one for the even coefficients, the other for the odd ones. The even part is nothing else than a 4 points DCT, again separated in even and odd parts in stage3. The round block in figure 14 signifies a multiplication by $1/\sqrt{2}$. In Fig.15, we present the butterfly block and the equations associated.



$$O_0 = I_0 + I_1$$
$$O_1 = I_0 - I_1$$

Fig. 15. The Butterfly block and its associated equations

The rectangular block depicts a rotation, which transforms a pair of inputs [I_0,I_1] into outputs [O_0,O_1]. The symbol and associated equations are depicted in Fig. 16.



$$O_0 = I_0 k \cos\left[\frac{n\pi}{16}\right] - I_1 k \sin\left[\frac{n\pi}{16}\right]$$
$$O_1 = -I_0 k \sin\left[\frac{n\pi}{16}\right] + I_1 k \cos\left[\frac{n\pi}{16}\right]$$

Fig. 16. The rotator block and its associated equations

The rotator block operation can be calculated using only 3 multiplications and 3 additions instead of 4 multiplications and 2 additions. This can be done by using the equivalence showed in the following equations.

$$\begin{aligned} O_0 &= a.I_0 + b.I_1 = (b-a).I_1 + a.(I_0 + I_1) \\ O_1 &= -b.I_0 + a.I_1 = -(b+a).I_0 + a.(I_0 + I_1) \end{aligned} \tag{5}$$

For the fast computation of two-dimensional (2-D) DCT/IDCT, there are two categories: row/column approach from 1-D DCT/IDCT [34]-[36] and direct 2-D DCT/IDCT [37]-[39]. However, the implementation of the direct 2-D DCT/IDCT requires much more effort and large area than that of the row/column approach [40]-[41] which is used to implement 2-D DCT/IDCT algorithms.

For the row/column approach, the 1-D DCT/IDCT of each row of input data is taken, and these intermediate values are transposed. Then, the 1-D DCT/IDCT of each row of the transposed values results in the 2-D DCT/IDCT. The modified Loeffler algorithm requires only 11 multiplications for the 8-point 1-D DCT/IDCT and 176 multiplications for the row/column 2-D DCT/IDCT.

3) Optimization in Quantization and Inverse Quantization: the quantization equations are not standardized in H.263 the ITU has suggested two quantizers in their Test model 8 (TMN8) [42] corresponding to INTRA and INTER modes and are given in (6)

$$LEVEL = \begin{cases} \dfrac{|COF|}{2.QP}, & INTRA \\ \dfrac{|COF| - \dfrac{QP}{2}}{2.QP}, & INTER \end{cases} \tag{6}$$

The INTRA DC coefficient is uniformly quantized with a quantized step of 8. The quantization parameter *QP* may take integer value from 1 to 31. *COF* stands for a transform coefficient to be quantized. *LEVEL* stands for the absolute value of the quantized version of the transform coefficient.

These equations are useful as a reference not only because they are commonly used as a reference model, but also because studies performed by the ITU during the standardization process [43] indicated that the quantization equations in (6) were nearly optimal subject to the constraints of uniform scalar quantization with a dead zone.

The basic inverse quantization reconstruction rule for all non-zero quantized coefficients is defined in equation 7 which give the relationship between coefficient levels (LEVEL), quantization parameter (QP) and reconstructed coefficients (REC)

$$|REC| = \begin{cases} QP.(2.|LEVEL| + 1), & if \quad QP = "odd" \\ QP.(2.|LEVEL| + 1) - 1, & if \quad QP = "even" \end{cases} \tag{7}$$

After calculation of $|\mathrm{REC}|$, the sign is added to obtain REC:

$$REC = sign(LEVEL).|REC| \tag{8}$$

The quantization and inverse quantization equations (6 and 7 respectively) are a regular formula and use multi-cycle to code data with NIOS II processor. To improve performance of our encoder, we can use single-cycle combinatorial NIOS II custom instruction logic to implement these equations. The custom instruction interface for quantization should be presented as in Fig. 17. As the processor need a 32-bit data interface. The defined interface are 32-bit input data (COF and QP which is fixed at 16 in our cas) and the output data (LEVEL).

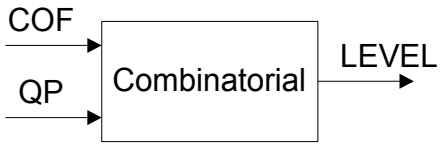


Fig. 17. Custom instruction interface for quantization

5. Design environment and FPGA implementation of H.263 encoder

A. Overview of the STRATIX II FPGA architecture

The Altera Stratix II EP2S60 FPGA is based on 1.2V, 90 nm technologies with a density that reaches 48352 Adaptive look-up tables (ALUTs), 310 KB of Embedded System Blocs (ESBs), 288 DSP blocks and 493 Input/Output Blocks (IOBs) [44]-[45].

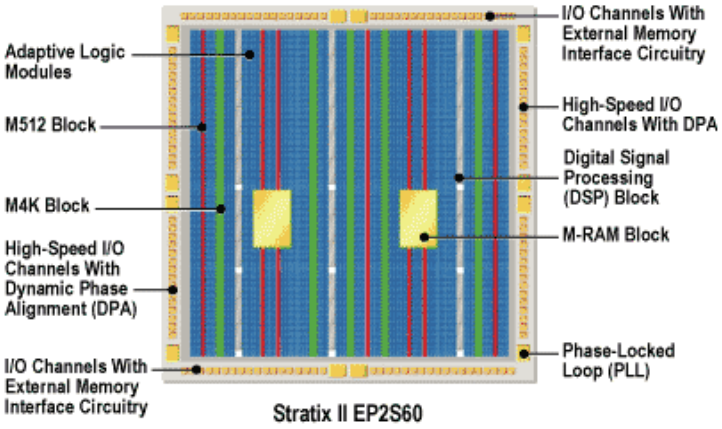


Fig. 18. Overview of Stratix II Die

An overview of the resources available in a Stratix II die is shown in Fig. 18. Three main advantages of this component led us to this choice. Firstly, Stratix II is optimized to maximize the performance benefits of SoPC integration based on NIOS II embedded processor. Secondly, Stratix II introduces DSP cores for signal processing applications. These embedded DSP Blocks have been optimized to implement several DSP functions with maximum performance and minimum logic resource utilization. The DSP blocks comprise a number of multipliers and adders. These can be configured in various widths to support multiply-add operations ranging from 9x9-bit to 36x36-bit, and including a wide range of operations from multiplication only, to sum of products, and complex arithmetic multiplication. Lastly, the Stratix II device incorporates a configurable internal memory called *TriMatrix* memory which is composed of three sizes of embedded RAM blocks. The Stratix II EP2S60 *TriMatrix* memory includes 329 M512 blocks (32x18-bit), 255 M4K blocks (128x36-bit) and 2 M-RAM (4Kx144-bit). Each of these blocks can be configured to support a wide range of features and to synthesize a wide variety of RAM (FIFO, double ports). With up to 310 KB of fast RAM, the *TriMatrix* memory structure is therefore appropriate for handling the bottlenecks arising in video embedded system.

B. FPGA Implementation of H.263 Video Encoder

The block diagram of the implemented H.263 encoder is shown in Fig.19. It is composed of three parts: a NIOS II softcore processor and 2-D DCT and 2-D IDCT hardware core. The main processing core of our system is the NIOS II CPU which is connected to hardware peripherals via a custom Altera's Avalon bus. The bus is a configurable bus architecture that is auto generated to fit the interconnection needs of the designer peripherals. The Avalon bus consists of the control, data and address signals and arbitration logic that are connected to the peripheral components.

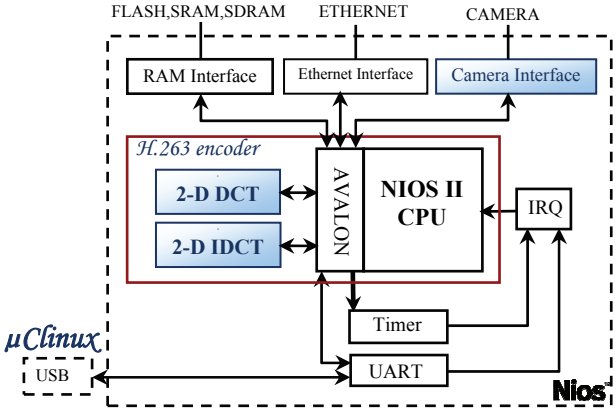


Fig. 19. Block diagram of the implemented H.263 encoder

Our system can receive frames from camera. For this purpose, we have developed a Camera interface for video acquisition [46]. The H.263 generated bit-stream has been downloaded through Ethernet Interface (FTP server) to PC platform in order to visualize the coded frames. Every hardware core is described in VHDL. Using Altera SOPC builder, the system was designed according to the block schematic diagram. The VHDL files were generated and the system was routed, compiled and downloaded into the FPGA using Altera Quartus II software. We have used the Modelsim™ simulator from Model Technology for circuit simulation.

1) System Environment: When the hardware is designed and fitted into a FPGA, there are two options how to port software applications on the board. The first is to use Linux operating system. µClinux is a port of the Linux operating system for embedded processors lacking a Memory Management Units (MMUs) [47]. Originally targeting the Motorola's 68K processor series, it now supports several architectures including NIOS II. The port of µClinux on the NIOS II core is licensed under the terms of the GNU General Public License (GPL) [48]. The second option is to use the monitor program which is loaded into the RAM of the NIOS II controller. This method is used during the development cycle. When the application meets the requirements, it is compiled for the Linux operating system.

2) 2-D DCT/IDCT coprocessor core: The 2-D DCT/IDCT transformation is implemented using the row/column approach which requires three steps: 8-point 1-D DCT/IDCT along the rows, a memory transposition and another 8-point DCT/IDCT along the transposed columns. Fig. 20 is a block diagram of the 2-D DCT/IDCT coprocessor core, showing the main interfaces and functional blocks.

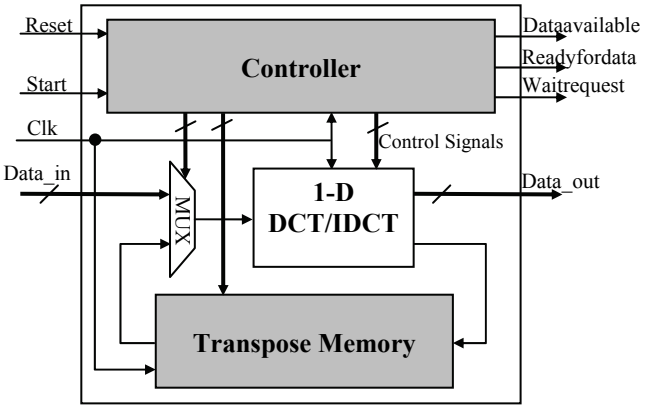


Fig. 20. 2-D DCT/IDCT coprocessor core

The controller is the control unit for the DCT/IDCT transformation. It receives input control signals (Reset, Start) and generates all the internal control signals for each stage and the output control signals for Avalon Bus communication (Dataavailable, Readyfordata, Waitrequest). When the Start signal is activated, the controller enables input of the first data row through Data_in signal. It then activates the 1-D DCT/IDCT unit for row data treatment. The first row of the transpose memory stores the results in an intermediate memory. This process repeats for the remaining seven rows of the input block. Next, the 1-D DCT/IDCT unit receives input data from the columns of the transpose memory under the MUX. The results of the column-wise 1-D DCT/IDCT are available through the Data_out signal.

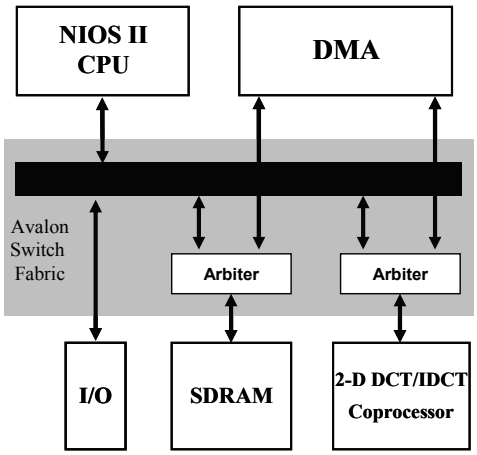


Fig. 21. Overview of the data path of a DMA with 2-D DCT/IDCT coprocessor and SDRAM

Data_in and Data_out signals are connected to the Avalon Bus. The 2-D DCT/IDCT coprocessor read/store the data from/to SDRAM through this Bus. Using processor to move data between SDRAM and 2-D DCT/IDCT coprocessor is less efficient. The system performance is greatly improved when the data are passed to the coprocessor with hardware. This is based on the concept of minimizing the interaction between the NIOS II

processor and the 2-D DCT/IDCT coprocessor. For better performance, data is handled by Direct Memory Access (DMA) as shown in Fig.21.
The 1-D DCT/IDCT unit based modified Loeffler algorithm which use 11 multipliers and 29 adders. In order to optimize speed and area of the 1-D DCT/IDCT implementation, we use Altera embedded DSP blocks to implement multipliers [49]. To conform to IEEE 1180-1990 accuracy specifications [50], the multiplier constants in Loeffler algorithm require a 12-bit representation. The DCT/IDCT use 24 internal registers to store intermediate values. The arithmetic units and registers use multiplexers to select inputs from internal and external registers. With these resources, a 1-D DCT/IDCT operation completes in 12 clock cycles and overall 2-D DCT/IDCT process concludes in 97 clock cycles.
The transpose memory is an internal memory of 64 words that holds the intermediate values from the first eight 1-D DCT/IDCT. The transpose memory receives input in a row-wise fashion and provides outputs in a column-wise fashion, thus performing a matrix transposition. Each row of the transposition memory is enabled for input from the 1-D DCT/IDCT unit after the first eight 1-D DCT/IDCT. For the next eight 1-D DCT/IDCT the column of the transposition memory output their data to the 1-D DCT/IDCT unit.

C. Implementation results

In table 1, implementation results of the H.263 encoder in Stratix II EP2S60 FPGA are shown.

	NIOS II (FAST)	2-D DCT coprocessor	2-D IDCT coprocessor
ALUTs	11%	3%	3%
ESBs	44%	1%	1%
DSPs	3%	8%	8%
IOBs	41%	15%	15%
Fmax (MHz)	227	133	139

Table 1. The implementation results in Stratix II FPGA

Results in the Table 1 have been obtained with separate implementation of the particular modules (NIOS II softcore processor, 2-D DCT and 2-D IDCT coprocessor core). The HW custom instruction for quantization and inverse quantization use only 1% of the ALUTs. The entire H.263 encoder utilizes 23% of the ALUTs, 44% of the ESBs, 18% of the DSP blocks and 41% of the IOBs. We can see that there is sufficient free space for other applications. The whole design works with a 120 MHz system clock. The implementation of H.263 encoder on the FPGA allows us to obtain a SoPC system.

6. Experimental results

The results discussed in this section are based on our HW/SW implementation of the H.263 which is tested on the Altera NIOS II development board. The results illustrate the tradeoffs among compression performance and coding speed. For all experiments the QCIF test sequences coded at 10frames/s with fixed quantization parameter QP=16. We focus on the following video test sequences: “Carphone”, “News”, “Claire”, and “Miss America”. These test sequences have different movement and camera operations. Carphone has frequent motion and camera movement. News has a combination of fast and slow motion which includes rotation movement and slow motion. Claire and Miss America have little motion with a stable camera.

	Software	Hardware	Speed up
2-D DCT	159881	720	222
2-D IDCT	159881	720	222
Q	4736	64	74
IQ	2560	64	40

Table 2. Clock cycles to code 8x8 block

Once the whole design are described in VHDL at the RTL level and fitted into the FPGA, we have determined coding time of H.263 encoder before and after timing optimization. The processor core clock and system clock are set to 120 MHz, thus 8.33 ns delay for each coded data is required. Table 2 shows a comparison of the clock cycles necessary to code an 8x8 block by software and hardware using the 2-D DCT, 2-D IDCT, Q and IQ.
Fig.22 presents a breakdown of the execution time before and after optimization of the H.263 encoder. The percentage distribution was very similar for all four sequences, so only the results for the Carphone and Miss America sequences are shown here. However, we can note that The HW/SW implementation of the H.263 provides a 15.8-16.5 times improvement in coding speed compared to software based solution.

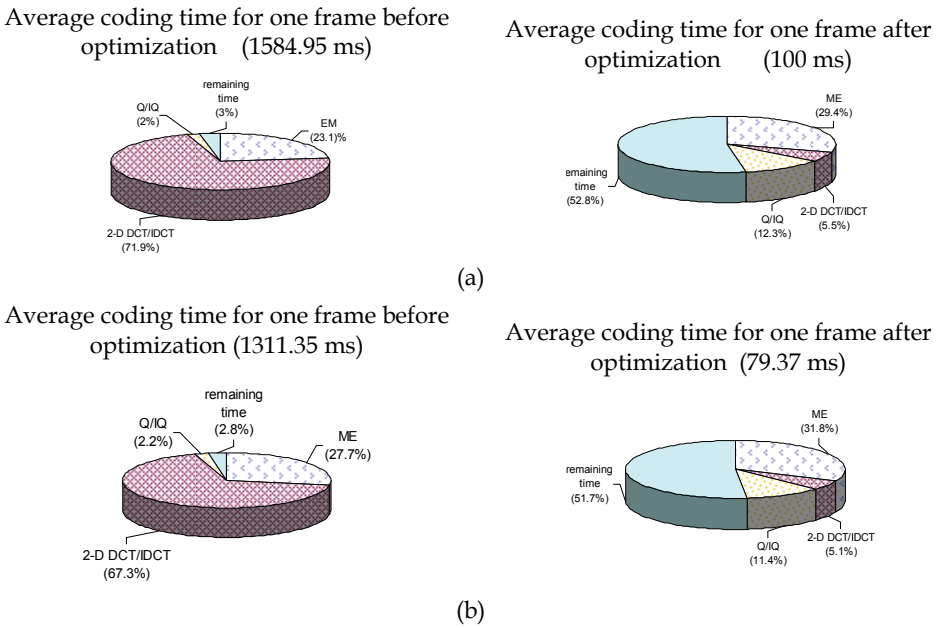


Fig. 22. CPU time percentage according to the processing before and after optimization of (a) Carphone and (b) Miss America sequences

The whole project was made under μClinux and performed on the NIOS II softcore processor. The H.263 generated bit-stream is send to the PC through Ethernet to analyse the results. The Fig.23 presents the original and the two reconstructed (one from SW, the other from HW/SW) of the 12th frame of the test video sequences. Also, Table 3 shows measurements of the PSNR of the luminance signal, Bit rate and coding speed.

Claire sequence	PSNR-Y = 33.67 dB	PSNR-Y = 33.44 dB
Miss America sequence	PSNR-Y = 35.2 dB	PSNR-Y = 34.95 dB
News sequence	PSNR-Y = 29.72 dB	PSNR-Y = 29.66 dB
Carphone sequence	PSNR-Y = 30.19 dB	PSNR-Y = 30.08 dB
(a)	(b)	(c)

Fig. 23. (a) Original, (b) Reconstructed from SW and (c) Reconstructed from HW/SW of the 12th frame of the test video sequences

Sequence	PSNR-Y (dB)	Bit Rate (Kbps)	Coding speed (fps)
Software Encoder			
Claire	33.67	8.22	0.78
Miss America	35.2	8.5	0.76
News	29.72	21.13	0.7
Carphone	30.19	29.82	0.63
HW/SW Encoder			
Claire	33.44	8.1	11.47
Miss America	34.95	8.44	12.6
News	29.66	21.35	10.94
Carphone	30.08	30.25	10

Table 3. Experimental results for HW/SW implementation of the H.263 video encoder

The quantities in Table 3 show the subjective visual impression that the image quality of the decompressed bit stream of the HW/SW encoder is nearly as good as it is with the output of the software encoder.
These results prove that after optimization our H.263 encoder can process 10-12.6 frames QCIF/sec which depend on the CPU clock frequency.

7. Conclusions

In this paper, we have described an efficient HW/SW codesign architecture of the H.263 video encoder into an embedded Linux environment. We have proposed timing optimization of the encoder. We have shown that a 15.8-16.5 times improvement in coding speed compared to software based solution can be obtained using the HW/SW implementation. We have presented a modern implementation method where the complex embedded system (H.263 encoder) can be efficiently HW/SW partitioned and optimized. Our architecture codes QCIF at 10-12.6 frames/sec with a 120 MHz system clock and can be improved with another FPGA platform having higher operating frequency.

8. References

[1] H. Li, A. Lundmark, and R. Forchheimer, "Image sequence coding at very low bitrates: A review," IEEE Trans. Image Processing, vol. 3, pp. 568–609, Sept. 1994.

[2] B. Girod, K. B. Younes, R. Bernstein, P. Eisert, N. Farber, F. Hartung, U. Horn, E. Steinbach, T. Wiegand, and K. Stuhlmuller, "Recent advances in video compression," in IEEE Int. Symp. Circuits Syst., Feb. 1996.

[3] B. Girod, "Advances in digital image communication," in Proc. 2nd Erlangen Symp., Erlangen, Germany, Apr. 1997.

[4] ITU-T Rec. H.263, Video Coding for Low Bit Rate communication. 1998.

[5] Y. li and Al "Hardware-Software Co-Design of Embedded Reconfigurable Architectures," Design Automation Conference 2000, Los Angeles, California.

[6] S. M. Akramullah, I. Ahmad and M. L. Liou. "Optimization of H.263 Video Encoding Using a Single Processor Computer: Performance Tradeoffs and Benchmarking," IEEE Trans. on Circuits and Syst. for Video Technology, vol. 11, pp. 901-915, Aug. 2001.

[7] K. -T. Shih, C.-Y. Tsai, H.-M. Hang, " Real-Time Implementation of H.263+ Using TMS320C6201 Digital Signal Processor," in Proc. IEEE ISCAS '03, vol. 2, pp. 900-903, May 2003.
[8] G. Lienhart, R. Lay, K. H. Noffz, R. Manner. "An FPGA-based video compressor for H.263 compatible bitstreams," in Proc. IEEE ICCE, pp. 320-321, June 2000.
[9] http://www.xilinx.com
[10] http://www.altera.com
[11] G. Côté, B. Erol, M. Gallant and F. Kossentini, "H.263+: Video Coding at Low Bit Rates," IEEE Trans. On Circuits And Systems. For Video Technology, vol. 8, pp. 849-866 , Nov. 1998.
[12] J. R. Jain and A. K. Jain "Displacement measurement and its applications in interframe image coding," IEEE Trans. on Communications, vol. 29, pp. 1799-1808, Dec. 1981.
[13] N. Ahmed, T. Natarajan and K. R. Rao, "On image processing and a discrete cosine transform," IEEE Trans, On Computers, vol. C-23,pp. 90-93, 1974.
[14] J. Johnston, N. Jayant, and R. Safranek, "Signal compression based on models of human perception," Proc. IEEE, vol. 81, pp. 1385-1422, Oct. 1993.
[15] R. Tessier and W. Burleson, "reconfigurable computing for digital signal processing: a survey," Journal of VLSI Signal Processing 28, 7-27, 2001
[16]Microblaze Integrated Development Environment http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze
[17] Nios II Integrated Development Environment http://www.altera.com/literature/lit-index.html
[18] Nios II Development Kit, Stratix II Edition, ALTERA 2006, http://www.altera.com/products/devkits/altera/kit-niosii-2S30.html
[19] SOPC Builder Applications ALTERA 2006, http://www.altera.com/products/software/products/sopc/sop-index.html
[20] J. Cong, Y. Fan, G. Han, A. Jagannathan, G. Reinman, Z. Zhang, "Instruction Set Extension with Shadow Registers for Configurable Processors," FPGA'05, February 20–22, 2005, Monterey, California, USA.
[21] Altera "NIOS Custom Instructions Tutorial", June 2002, http://www.altera.com/literature/tt/tt_nios_ci.pdf
[22] C. Zhu, X. Lin, and L. P. Chau, "Hexagon-Based Search Pattern for Fast Block Motion Estimation", IEEE Trans. On Circuits And Systs. For Video Technology, vol. 12, pp. 349-355, May 2002
[23] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 8, pp. 369–377, Aug. 1998.
[24] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast blockmatching motion estimation," IEEE Trans. Image Process., vol. 9, no. 2, pp. 287–290, Feb. 2000.
[25] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," IEEE Trans. Circuits Syst. Video Technol., vol. 6, no. 4, pp. 419–423, Aug. 1996.
[26] C. H. Cheung and L. M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 12, no. 12, pp. 1168–1177, Dec. 2002.

[27] Y.P Lee and all "A cost effective architecture for 8x8 two-dimensional DCT/IDCT using direct method," IEEE Trans. On Circuit and System for video technology, VOL 7, NO.3, 1997

[28] W.c Chen, C.h Smith and S.C. Fralick, "A fast Computational Algorithm for thr Discrete Cosine Transform," IEEE Trans. On Communications, Vol. COM-25, No. 9, pp.1004-1009, Sept.1997

[29] C. Loeffler and A. Lightenberg, "Practical fast 1-D DCT algorithms with 11 Multiplications," in Proceedings IEEE ICASSP '89, vol. 2, pp. 988-991, May 1989.

[30] T.J. van Eijndhven and F.W. Sijstermans, "Data Processing Device and method of Computing the Cosine Transform of a Mtrix", PCT Patent WO 99948025, to Koninklijke Philips Electronics, World Intellectual Property Organization, International Bureau, 1999.

[31] P. Duhamel and H. H'Mida, "New 2^n DCT algorithms suitable for VLSI implementation," in Proc. ICASSP'87, vol. 12, pp. 1805–1808, Apr. 1978.

[32] M. T. Heidemann, "Multiplicative Complexity, Convolution, and the DFT," New York: Springer-Verlag, 1988.

[33] E. Feig and S. Winograd, "On the multiplicative complexity of discrete cosine transforms," IEEE Trans. Inform. Theory, vol. 38, pp. 1387–1391, July 1992.

[34] M. D. Wagh and H. Ganesh, "A new algorithm for the discrete cosine transform of arbitrary number of points," IEEE Trans. Comput., vol. C-29, pp. 269–277, Apr. 1980.

[35] B. G. Lee, "A new algorithm to compute the discrete cosine transform," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-35, pp. 1243–1245, Dec. 1984.

[36] Y. Chan and W. Siu, "A cyclic correlated structure for the realization of discrete cosine transform," IEEE Trans. Circuits Syst.–II, vol. 39, pp. 109–113, Feb. 1992.

[37] M. Vetterli, "Fast 2-D discrete cosine transform," in Proc. IEEE ICASSP'85, vol. 10, pp. 1538–1541, Mar. 1985.

[38] N. I. Cho and S. U. Lee, "DCT algorithms for VLSI parallel implementation," IEEE Trans. Acoust., Speech, Signal Processing, vol. 38, pp. 121–127, Jan. 1990.

[39] N.I. Cho, S.U. Lee, "A fast 4 _ 4 DCT algorithm for the recursive 2-D DCT," IEEE Trans. Signal Processing, vol. 40, pp. 2166–2173, Sept. 1992.

[40] C. Y. Lu, K. A. Wen, "On the design of selective coefficient DCT module", IEEE Trans. Circuits Syst. Video Technol., vol. 8, pp. 143–146, Dec. 2002.

[41] J. Liang, "Fast multiplierless approximations of the DCT with the lifting scheme", IEEE Trans. Signal Process., vol. 49, pp. 3032–3044, Dec. 2001.

[42] ITU Telecom. Standardization Sector of ITU, "Video codec test model near-term, Version 8 (TMN8), Release 0," H.263 Ad Hoc Group, June 1997.

[43] Proposal for Test Model Quantization Description, ITU-T doc. Q15-D-30, Apr. 1998.

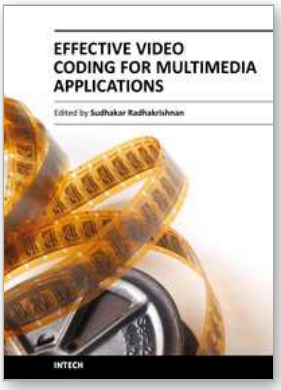
[44] D.Lewis and Al, "The Stratix II Logic and Routing Architecture," FPGA'05, February 20–22, 2005, Monterey, California, USA.

[45] Altera Startix II Architecture
http://www.altera.com/products/devices/stratix2/st2-index.jsp

[46] A. Ben Atitallah, P. Kadionik, F. Ghozzi, P.Nouel, N. Masmoudi, Ph.Marchegay "Hardware Platform Design for Real-Time Video Applications," in Proc. IEEE ICM'04, pp. 722-725, Dec. 2004.

[47] The μClinux project http://www.uClinux.org.

[48] The NIOS Forum http://www.niosforum.com/forum.
[49] A. Ben Atitallah, P. Kadionik, F. Ghozzi, P. Nouel, "Optimization and implementation on FPGA of the DCT/IDCT algorithm", in Proc. IEEE ICASSP'06, vol. 3, pp. 928-931, May 2006.
[50] IEEE Std 1180-1990, "IEEE standard specification for the implementation of 8x8 inverse cosine transform," Institute of Electrical and Electronics Engineers, New York, USA, International Standard, Dec. 1990



Effective Video Coding for Multimedia Applications

Edited by Dr Sudhakar Radhakrishnan

ISBN 978-953-307-177-0
Hard cover, 292 pages
Publisher InTech
Published online 26, April, 2011
Published in print edition April, 2011

Information has become one of the most valuable assets in the modern era. Within the last 5-10 years, the demand for multimedia applications has increased enormously. Like many other recent developments, the materialization of image and video encoding is due to the contribution from major areas like good network access, good amount of fast processors e.t.c. Many standardization procedures were carrried out for the development of image and video coding. The advancement of computer storage technology continues at a rapid pace as a means of reducing storage requirements of an image and video as most situation warrants. Thus, the science of digital video compression/coding has emerged. This storage capacity seems to be more impressive when it is realized that the intent is to deliver very high quality video to the end user with as few visible artifacts as possible. Current methods of video compression such as Moving Pictures Experts Group (MPEG) standard provide good performance in terms of retaining video quality while reducing the storage requirements. Many books are available for video coding fundamentals.This book is the research outcome of various Researchers and Professors who have contributed a might in this field. This book suits researchers doing their research in the area of video coding.The understanding of fundamentals of video coding is essential for the reader before reading this book. The book revolves around three different challenges namely (i) Coding strategies (coding efficiency and computational complexity), (ii) Video compression and (iii) Error resilience. The complete efficient video system depends upon source coding, proper inter and intra frame coding, emerging newer transform, quantization techniques and proper error concealment.The book gives the solution of all the challenges and is available in different sections.

How to reference

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

A. Ben Atitallah, P. Kadionik, F. Ghozzi, P.Nouel, N. Masmoudi and H. Levi (2011). An FPGA Implementation of HW/SW Codesign Architecture for H.263 Video Coding, Effective Video Coding for Multimedia Applications, Dr Sudhakar Radhakrishnan (Ed.), ISBN: 978-953-307-177-0, InTech, Available from: http://www.intechopen.com/books/effective-video-coding-for-multimedia-applications/an-fpga-implementation-of-hw-sw-codesign-architecture-for-h-263-video-coding

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai

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

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