A Padding Processor for MPEG-4

Georgi Kuzmanov, Stamatis Vassiliadis, Jos van Eijndhoven*

Delft University of Technology - Electrical Engineering Dept.,
P.O. Box 5031, 2600 GA Delft,The Netherlands
*PHILIPS Research - Dept. of Information and Software Technology,
Eindhoven,The Netherlands

Phone:+31-(0)15-27-86249 E-mail:G.Kuzmanov@ET.TUDelft.NL

Abstract— **This paper proposes a new processing unit, which performs the MPEG-4 padding algorithm in real time. The padding unit has been implemented as a scalable systolic structure of *processing elements*. We have written a synthesizable VHDL model of a single processing element and have run RTL simulations. A generic array of PE has been described in VHDL as well, and the functionality of the unit has been validated by simulations. In order to determine the actual parameters (chip area and speed) of the padding processor, we have synthesized the VHDL model for two FPGA families - Xilinx and Altera. Three configurations of the array have been investigated for both xc4085xlpg559-09 and epf10k20rc240-4 chips. These configurations consist of 4, 8, and 16 PEs, which have been evaluated in terms of FPGA area and operating frequency. The simulation results indicate that the proposed padding unit can operate in the frequency range 11,4-24,5 MHz and the real-time requirements can be easily met at a reasonable hardware cost. Finally, the trade-off between chip-area and operating speed is discussed and possible configuration alternatives are proposed. The proposed padding engine can be implemented as a part of content-based encoders (e.g., MPEG-4 and MPEG-7), either as a hardwired unit, or as a reconfigurable engine in a Custom Computing Machine.**

Keywords— **MPEG-4, Repetitive Padding, Visual Object Plane, Systolic Structure**

I. Introduction

The inclusion of entirely new content-based functionalities in MPEG-4 [5], [6] makes most of the specialists refer to it as a new standard generation, rather than the next MPEG version. Essentially, MPEG-4 is the first standard to deal with content-based coding. For content-based coding, MPEG-4 uses the concept of a Video Object Plane (VOP). A VOP is an arbitrarily shaped region of a frame, which usually corresponds to a semantic object in the visual scene. A sequence of VOPs in the time domain is referred to as a Video Object (VO). Each VOP is described by its *shape* and *texture*. Shape is mainly represented in binary format. This format represents the shape as a bitmap, referred to as *binary alpha plane*. Each pixel in this plane takes one of two possible values, which indicate whether the pixel belongs to the object or not. The binary alpha plane is divided into 16x16-pixel blocks called *Binary Alpha Blocks (BAB)*. The texture of a VOP represents its color by *macroblocks*. Each macroblock consists of one 16x16 array of luminance (grayscale) pixels and two 8x8 arrays of chrominance (color) pixels, which represent the full-color of the corresponding 16x16 area of a VOP. Like its predeceding visual data compression standards, MPEG-4 adopts *motion compensation* techniques to exploit temporal redundancies in the encoded video sequences. Motion compensation is a process of coding differences (motion) between frames in a video sequence [9]. These differences are estimated as a displacement between pixel areas in the current frame (being encoded) and a previously encoded frame. The measurement of this displacement is the *motion vector*. A process, called *motion estimation*, is performed to determine the motion vectors for each macroblock. This process includes a search algorithm for best matching between the block to be encoded and an area of previously encoded frame. One important new feature in MPEG-4 is the *padding* technique. *The purpose of padding in MPEG-4 is to ensure more accurate block matching in motion compensation algorithms for arbitrary shaped visual objects.* Profiling results, reported in [3], [4], [10], [11], indicate that padding is a computationally demanding and time consuming process, which restricts the real time operation of the MPEG-4 codecs.

In this paper we propose a scalable padding engine, capable to perform the padding algorithm in real time. An operating frequency in the range of 11.4-24.5 MHz is achieved by implementations, mapped on regular FPGA chips (Altera and Xilinx). This frequencies are sufficient for real time processing of macroblocks

in the most demanding profiles (core profile and main profile) of MPEG-4 [7].

The remainder of the discussion is organized as follows. Section II describes the repetitive padding algorithm. In Section III, the design of the padding unit is proposed. Section IV discusses the simulation results. Finally, the conclusions are represented in Section V.

II. The Repetitive Padding Algorithm

For motion compensation/decompensation of VOPs, MPEG-4 adopts the padding process, which aims at more accurate block matching. This process defines the full-color values (luminance+chrominance) for pixels outside the shape of a VOP. In padding, two types of macroblocks are of interest. Macroblocks, which lie on the boundary of the VOP are referred to as boundary blocks. They are processed by the so called *repetitive padding*. Exterior macroblocks (completely outside the VOP) are padded using the *extended padding method.* Since repetitive padding is the most demanding padding algorithm, in this paper we will consider the padding of boundary macroblocks. The repetitive padding algorithm is described in [5], [9], but in the literature some modifications can be met. In all these modifications, however, the boundary block is separately processed horizontally, per scan-line basis and vertically - per columns. In [2] the padding algorithm is modified to support specific instruction set extensions. In the same paper, the horizontal and vertical padding are divided into two phases each. These two phases consequently scan the lines/columns into two opposite directions and perform the padding operations. In this paper we are using the standard repetitive padding algorithm. The repetitive padding algorithm, as defined in [5], [9], has the following steps:

1. Define any pixel outside the object boundary as a zero pixel. Make a duplicate binary alpha map.
2. Scan each horizontal line of a block. Each scan line is possibly composed of *zero* and *nonzero* line segments (according to the shape bits in the binary alpha map).

(a) In zero segments, between an end point of the scan line and the end point of a nonzero segment, all zero pixels are replaced by the pixel value of the end pixel of nonzero segment.

(b) In zero segments, between the end points of two different nonzero segments, all zero pixels take the average value of these two end points.

Nonzero segments are not processed. All shape bits, corresponding to padded pixels are set in the duplicate binary alpha map.

3. Scan each vertical line of the block and perform the identical procedure as described for the horizontal line. The updated shape information from the duplicate binary alpha map is used.

III. The Processing Unit

Since padding is performed over horizontal and vertical pixel lines in identical manner, we propose a scalable systolic structure to process pixel blocks per line basis. Therefore we propose the processing element (PE), depicted in Figure 1, which is dedicated to process each pixel of a block. The same processing element is used for luminance and chrominance padding. The following equations describe the functionality of the processing element:

$|O|_{\overline{N}} = S.|I|_{\overline{N}} \vee \overline{S}.\{(|LI|_{\overline{N}} + |RI|_{\overline{N}}) >> i\}$,

$|LO| = S.|I| \vee \overline{S}.|RI|$,

$|RO| = S.|I| \vee \overline{S}.|LI|$,

$S' = S \vee LI_N \vee RI_N$;

where $i = LI_N \wedge RI_N$,

N represents the width of the processed data (default N=8),

$|LI|, |RI|$ are left and right input vectors with width N+1,

$|LO|, |RO|$ are left and right output vectors with width N+1,

$|I|, |O|$ are data input and output vectors with width N,

S is the shape (input) bit before processing,

S' is a mask output bit after processing,

$|LI|_{\overline{N}}$ denotes the first N bits of LI (bits 0 to N-1), and $|LI|_N$ represents the N^{th} bit of LI.

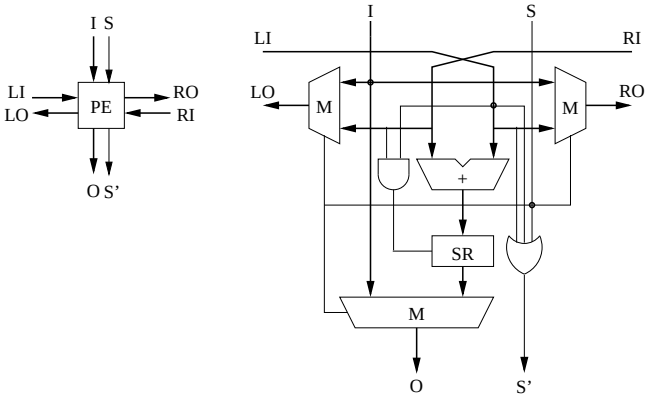

Fig. 1. Padding Processing Element

The operation of the PE is as follows. If the input shape bit S is set (the pixel belongs to the object), then:

1. The output O takes the value of the input I, i.e. the pixel keeps its color.
2. The value of the input (pixel) I is propagated to the left and to the right (via outputs $LO_{\overline{N}}$ and $RO_{\overline{N}}$) for further processing. The shape input bit S is propagated by the same multiplexers and occupies the most-significant bits of LO and RO.
3. The output bit S' is set, meaning the pixel has been processed.

If the input shape bit S is zero (the pixel does not belong to the object and has to be padded), then:

1. The output O takes the average value of the $LI_{\overline{N}}$ and $RI_{\overline{N}}$ inputs, i.e. the pixel takes the padded value.
2. The LI value is propagated via RO and the RI - via LO including color and shape information.
3. The output bit S' is set, meaning the pixel has been processed.

To process a line from a macroblock, we implement the systolic structure of processing elements, depicted in Figure 2. For the proper circuit operation, the left-most and right-most inputs of the structure should be initialized with zero vectors. This would mean that there are no pixels to the left and to the right of the macroblock, which could influence the padding values. This structure is scalable and can contain an arbitrary number of processing elements. Since a macroblock consist of one 16x16 luminance and two 8x8 chrominance blocks, it is efficient to implement structures of 8 or 16 PE. Furthermore, it is possible to implement several structures, identical to the one in Figure 2. For example, if we implement eight such structures, we will be able to process eight lines in parallel. This is possible, because there is no data dependency between any two lines. The data dependency is just between the pixels in the same line. Even more complicated, two dimensional structure can be implemented, for processing a whole block in parallel. Implementations, which process more than one line in a time, however, require higher data throughput and the implementation of a two-dimensional block addressing would become necessary [8]. In this paper we evaluate only the single line/column implementation of the padding unit, depicted in Figure 2.

The proposed processing unit can be implemented either as a hardwired accelerator in a dedicated MPEG-4 codec circuit, or as a reconfigurable accelerator in a Custom Computing Machine [12]. In this paper, we evaluate the implementation of the padding unit as a reconfigurable processor, mapped onto an FPGA chip.

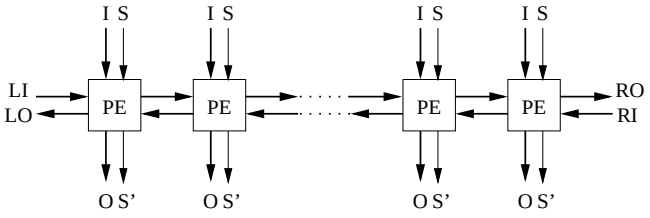

Fig. 2. Single Scan Line/Column Padding Structure

IV. Simulation Results

To evaluate the proposed structure, we have written a synthesizable VHDL model of a single PE and we have explored the models of configurations with different numbers of PEs. The evaluation has been made in terms of *chip area* and *speed*. To get realistic values for these two parameters, we have synthesized the VHDL models for two popular FPGA families - Altera and Xilinx, using their dedicated synthesis and simulation tools. We have not chosen the cutting edge of technology chips for the implementation, because we have been interested in achieving high performance with lower technological generation (hence cheaper) FPGAs. The Xilinx xc4085xlpg559-09 chip is clocked at a global speed of 96 MHz [13] and the Altera epf10k20rc240-4 chip - at 100 MHz [1]. Obviously, these two chips can be run at comparable speeds. Their chip area is quite different, but since area is estimated in different units for each of these companies, a comparison between the implementations on different chip families has not been performed.

For both chip families we have evaluated structures of 4, 8 and 16 PEs and speed has been measured in MHz. Table I suggests the area estimates for the Xilinx chip in absolute units - CLBs (Configurable Logic Blocks) and in percentage of the available gate array area. For the Altera chip, results are reported in Table II in similar manner but the absolute units are referred to as LCs (Logical Cells).

TABLE I
Results for the Xilinx xc4085xlpg559-09 chip

# PE	# CLB's		Speed
	total	%	MHz
16	419 of 3136	14	11.4
8	206 of 3136	7	18.2
4	45 of 3136	4	24.5

The speed estimations for both FPGA families suggest similar results. The reported numbers indicate that the padding structure can meet the real-time re-

TABLE II
RESULTS FOR THE ALTERA EPF10K20RC240-4 CHIP

# PE	# LC's		Speed
	total	%	MHz
16	1024 of 1152	88	13.4
8	511 of 1152	44	19.8
4	254 of 1152	22	24.8

quirements for a broad range of visual resolutions. If we consider the implementation wit 16 PEs, the estimated operating frequencies (11.4 MHz and 13.4 MHz) mean that the padding unit can process up to 237 500 MB/s (macroblocks per second) or 279 200 MB/s, depending on the FPGA family. For a QCIF format, a VOP as large as the whole frame has 396 macroblocks. To process such a VOP at a real time rate of 25 VOP/s we need a processing rate of 9900 MB/s. These numbers indicate that we can process at least 24 QCIF VOP/s in the worst case. In MPEG-4, however, padding is performed over boundary macroblocks, which are statistically 30% of the whole number of macroblocks in a VOP. This means that we can process approximately 80 VOP/s in a QCIF format with the achieved processing speed of the padding unit. For higher resolutions the number of the processed VOPs will be lower, but even then a sufficient number of processed VOPs per second can be achieved. Since VOPs may vary in size and resolution, the MPEG-4 requirements group has defined the binding criteria for implementation complexity in terms of *transferred macroblocks per second*. For the core and main MPEG-4 profiles, the required processing speeds are 23860 MB/s and 97200 MB/s for 16 and 32 VOPs respectively [7]. It is obvious that the operating speed, achievable by the proposed padding unit, is far over these values as well.

Results in Table I and Table II also indicate that the actual operating speed grows slower than the area increase rate. For example, the 4 times smaller implementation (in terms of chip area) is just around a factor of two slower then the implementation with 16 PEs. This property of the implementation can be used when either the area or the speed constrains of the unit are crucial. For the simulated single line/column repetitive padding processor, there could be several configuration options. Some of them are:
1. 16PE unit - processes one luminance line/column and two chrominance per operating cycle.
2. 8PE or 4PE unit - processes a half/quarter of luminance and one/half chrominance line/column.
3. 32, 64, ..., 256PE unit- processing more than two luminance and more than four chrominance lines/columns per operating cycle. The extreme configuration would process the whole macroblock.

V. CONCLUSIONS

In this paper we proposed a scalable padding engine, capable to process macroblocks in MPEG-4. The unit has been modeled in VHDL and its performance and hardware costs have been evaluated for two FPGA families - Altera and Xilinx. The simulation results indicate that the proposed padding unit can easily meet the real-time requirements of the core and main MPEG-4 profiles at a reasonable hardware cost. An operating frequency of up to 13.4MHz allowed a processing speed of up to 279 200 MB/s to be achieved by reasonably cheap FPGA chips. The proposed padding engine can be implemented either as a hardwired, or as a reconfigurable accelerator of content-based codecs (e.g., MPEG-4 and MPEG-7).

VI. ACKNOWLEDGEMENTS

This research is supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs, the Technology Foundation STW (project AES.5021) and PHILIPS Research Laboratories, Eindhoven, The Netherlands.

REFERENCES

[1] ALTERA. *Data Book*. Altera Corp., 1998.
[2] M. Berekovic, H.-J. Stolberg, M. B. Kulaczewski, P. Pirsh, H. Moler, H. Runge, J. Kneip, and B. Stabernack. Instruction set extensions for mpeg-4 video. *Journal of VLSI Signal Processing*, 23(1):27–49, October 1999.
[3] H.-C. Chang, L.-G. Chen, M.-Y. Hsu, and Y.-C. Chang. Performance analysis and architecture evaluation of MPEG-4 video codec system. In *IEEE International Symposium on Circuits and Systems*, volume II, pages 449–452, Geneva, Switzerland, 28-31 May 2000.
[4] H.-C. Chang, Y.-C. Wang, M.-Y. Hsu, and L.-G. Chen. Efficient algorithms and architectures for MPEG-4 object-based video coding. In *IEEE Workshop on Signal Processing Systems*, pages 13–22, 11-13 Oct 2000.
[5] ISO/IEC JTC11/SC29/WG11, N3312. MPEG-4 video verification model version 16.0.
[6] ISO/IEC JTC11/SC29/WG11 N4030. MPEG-4 overview, March. 2001.
[7] ISO/IEC JTC11/SC29/WG11 W2502. ISO/IEC 14496-2. Final Draft International Standard. Part2: Visual, Oct. 1998.
[8] G. Kuzmanov, S. Vassiliadis, and J. van Eijndhoven. MPEG-4 Addressing and ACQ Function. In *Second Workshop on Embedded Systems PROGRESS 2001*, Veldhoven, The Netherlands, 18 October 2001.

[9] Y. Q. Shi and H. Sun. *Image and Video Compression for Multimedia Engineering.* Boca Raton CRC Press, 2000.

[10] H.-J. Stolberg, M. Berekovic, P. Pirsch, H. Runge, H. Moller, and J. Kneip. The M-PIRE MPEG-4 codec DSP and its macroblock engine. In *IEEE International Symposium on Circuits and Systems*, volume II, pages 192–195, Geneva, Switzerland, 28-31 May 2000.

[11] S. Vassiliadis, G. Kuzmanov, and S. Wong. MPEG-4 and the New Multimedia Architectural Challenges. In *15th International Conference SAER'2001*, St.Konstantin, Bulgaria, 21-23 Sept. 2001.

[12] S. Vassiliadis, S. Wong, and S. Cotofana. The MOLEN rm-coded processor. In *11th International Conference on Field Programmable Logic and Applications (FPL)*, 2001.

[13] XILINX. *DataSource CD-ROM.* XILINX, 2000.