Reconfigurable Repetitive Padding Unit

Georgi Kuzmanov
G.Kuzmanov@ET.TUDelft.NL

Stamatis Vassiliadis
S.Vassiliadis@ET.TUDelft.NL

Computer Engineering Lab, Electrical Engineering Department,
Faculty of Information Technology and Systems, Delft University of Technology,
P.O. Box 5031, 2600 GA Delft, The Netherlands

ABSTRACT

This paper proposes a reconfigurable processing unit, which performs the MPEG-4 repetitive padding algorithm in real time. The padding unit has been implemented as a scalable systolic structure of *processing elements*. A generic array of PE has been described in VHDL, and the functionality of the unit has been validated by simulations. In order to determine the chip area and speed of the padding structure, we have synthesized the structure for two FPGA families - Xilinx and Altera. The simulation results indicate that the proposed padding unit can operate in a wide frequency range, depending on the implemented configuration. It is shown that it can process from tens up to hundreds of thousands MPEG-4 macroblocks per second. This allows the real-time requirements of all MPEG-4 profiles and levels to be met efficiently at trivial hardware costs. Finally, the trade-off between chip-area and operating speed is discussed and possible configuration alternatives are proposed.

Categories and Subject Descriptors

B.6.1 [**Logic Design**]: Design Styles—*logic arrays; parallel circuits*; B.5.1 [**RTL Implementation**]: Design—*arithmetic and logic units; data-path design; styles*

General Terms

Design, Performance

1. INTRODUCTION

The first versions of the MPEG standards, MPEG-1 and MPEG-2, aimed at providing best visual quality at given bitrate ranges. This concept has been preserved for the next standard versions as well. However, the inclusion of entirely new content-based functionalities in MPEG-4 [8] has made most of the specialists refer to it as a new standard generation, rather than the next MPEG version. To allow the efficient implementation of the standard, the MPEG-4 requirements define several application profiles. These profiles group the large set of required tools, according to the targeted classes of applications. Within each profile, a number of levels constrain the computational complexity and the required data bandwidth of the application. Each profile level states the values for certain parameters, which are used to judge whether an application meets the functional and implementational requirements of the level.

In literature [10], complexity analyzis results indicate that the computational requirements of the highest profiles and levels of MPEG-4 are measured in billions of RISC-like operations per second. These numbers will significantly exceed the capabilities of the general purpose processors, despite the near future technology achievements. The work presented in this paper is a part of a research activity, in which we focus on speeding up MPEG-4 applications. Our basic approach is to extend general-purpose architectures with application specific co-processing mapped on FPGAs and the use of microcode to improve FPGA setting and execution of instructions [15]. Due to the different requirements of the MPEG-4 profiles and the requirements for cost-effective implementations, a demand for high flexibility is essential. Therefore we believe that *reconfigurable* (FPGA) implementations of application specific functions would be beneficial for improving the cost/performance ratio of the whole system. One important new feature in MPEG-4 is the *padding* technique, defined at all Levels in the Core and Main Profiles of the standard. Software profiling results, reported in [3, 10, 13, 14], 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 unit, capable to perform the padding algorithm in real time. In principle, the unit can be implemented either as a hardwired circuit in a dedicated MPEG-4 codec, or as a reconfigurable accelerator in a Custom Computing Machine. For this paper, we focused the reconfigurable implementation of the padding unit, mapped onto FPGA chips and we have achieved the following:

- Real time processing for all MPEG-4 profiles and levels, utilizing the padding algorithm can be achieved.
- Scalable implementation, tunable to the different Profiles@Levels requirements. A worst-case scenario allow data processing rates from 77 K up to 280 K macroblocks/s and we show that even much higher speeds are achievable.
- Reconfigurable (FPGA) implementations require trivial hardware costs (low number of reconfigurable cells - 419 Xilinx CLBs and 1024 Altera LCs for a 16-pixel line processing) and low frequencies (varying between 11 and 25 MHz)

GLSVLSI'02, April 18-19, 2002, New York, New York, USA.

All these design achievements have been validated both for Altera and Xilinx FPGA families. The reconfigurability and scalability of the implementation allow higher flexibility in tuning the unit for optimal performance of the processor.

The remainder of the discussion in this paper is organized as follows. Section 2 gives some background information and the motivation for the presented research including a description of the repetitive padding algorithm. In Section 3, the design of the padding unit is proposed. Section 4 discusses some simulation results and evaluates the unit. Finally, the conclusions are presented in Section 5.

2. BACKGROUND AND MOTIVATION

Generally speaking, MPEG-4 is the first standard to deal with content-based coding. For content-based coding, the concept of a Video Object Plane (VOP) is used [7]. 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.

As its preceding 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 [12]. 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. In MPEG-4 the process is defined over VOPs instead of frames.

The Repetitive Padding Algorithm For more accurate block matching in motion compensation/ decompensation of VOPs, MPEG-4 adopts the padding process. *The purpose of padding in MPEG-4 is to ensure more accurate block matching in motion compensation algorithms for arbitrary shaped visual objects.* The padding 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 standard repetitive padding algorithm, as defined in [7], 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.

Figure 1 illustrates the repetitive padding algorithm with a simplified examples of a 4x4 pixel BAB and a 4x4 pixel luminance (chrominance) block. The original data structures (in the left of the Figure) are padded vertically and horizontally and both the intermediate and final results are indicated in this example.

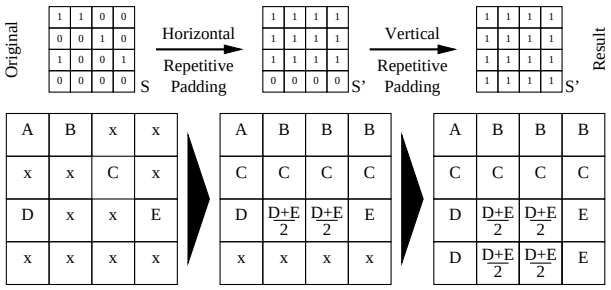


Figure 1: The Repetitive Padding Algorithm

Motivation. Unlike its predecessors, MPEG-4 is much more demanding in terms of computational complexity with even more data intensive algorithms. This is illustrated in Table 1, which represents the required data processing speed according to the MPEG-4 Visual Profiles@Levels definitions [6]. We can consider that the performance demands of the Simple MPEG-4 Profile are approximately the same as of MPEG-2. Therefore, the challenge is to meet the requirements of the most-demanding Core and Main Visual Profile Levels of MPEG-4. The software versus hardware implementation discussion has been emerged during the standardization process of MPEG-4. A summary of the computational complexity of the QCIF, Core Profile Level 1 of MPEG-4 is reported in [10]. Since this is the lowest profile level, utilizing the padding algorithm, we shall consider its real-time requirements as the minimum for a hardware implementation. At this level, the computational power, reported for the software encoding of a single object [10] is in the order of 4500 Million RISC-like Instructions Per Second (MIPS). Assuming a software performance optimization by

Table 1: MPEG-4 Visual Profiles@Levels Definitions and Required Data Processing Speed (in MacroBlocks per second [MB/s])

Profile	Level	Session Size	# VO	Max. MB/s	Boundary MB/s
Main	L4	1920x1088	32	489600	244800
	L3	CCIR 601	32	97200	48600
	L2	CIF	16	23760	11880
	L1	N.A.	N.A.	N.A.	N.A.
Core	L2	CIF	16	23760	11880
	L1	QCIF	4	5940	2970
Simple Scalable	L2	CIF	4	23760	N.A.
	L1	CIF	4	7425	N.A.
Simple	L3	CIF	4	11880	N.A.
	L2	CIF	4	5940	N.A.
	L1	QCIF	4	1485	N.A.

a factor of up to 10, the total computational complexity is just within the computational capabilities of the contemporary general purpose processors(500-1000 MIPS). In the case of 4 video objects (see Table 1), however, the real-time software feasibility becomes problematic. Therefore, the need of a hardware acceleration of MPEG-4 is evident, even at this relatively low profile level. Further analyzis of the requirements for the software implementation indicates that the padding algorithm occupies some 175 MIPS for a single video object, or around 700 MIPS for the maximum 4 video objects, stated at Level 1 of the Core profile (Table 1). A general purpose processor with such computational power is still very expensive to be dedicated just for the repetitive padding calculations. Considering Table 1, we can estimate that the required speed of 5940 MB/s for the Core Profile Level 1 is approximately 82 times lower than the speed requirements of the highest - Main@Level4 Profile (489600 MB/s). A simple arithmetic estimation indicates that for the highest MPEG-4 profile level, the non-optimized software padding would require approximately 57 000 MIPS and when extremely optimized (10 times speed-up) - in the order of 6000 MIPS. Even for the significantly less complex decoder part of MPEG-4, the padding algorithm will require some 24 000 MIPS for non-optimized software implementation down to 2500 MIPS in dramatically optimized programming. These numbers, large even for the expected technology levels of the near future, motivated our research to focus on a cost-effective hardware solution of the MPEG-4 algorithms and the repetitive padding in particular.

The large number of alternatively used algorithms in MPEG-4, however, makes the implementation of dedicated hardwired units inefficient, since a number of them may remain unutilized in certain profile levels. A good example is the padding algorithm, which is not included in the Simple Profile Levels of MPEG-4. Thus, a multi-profile codec would not utilize a hardwired padding accelerator, when running at any of the Simple Profile levels. A promising solution of this problem is the reconfigurable implementation of hardware accelerators, like the reconfigurable padding unit described in details in the section to follow.

Related work. Although the repetitive padding algorithm is described in [7, 12], in the literature some modifications of it are also reported. In [4, 9, 11] new algorithms or algorithm modifications are proposed to substitute the original repetitive padding or to redefine it. In [2] the original MPEG-4 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. A relatively complicated hardware padding unit is proposed in [5]. In the present paper we propose a simpler reconfigurable and scalable solution (differenciates from [5]), using the standard repetitive padding algorithm (differenciates from [2, 4, 9, 11]).

3. 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. Consequently, we need to define an elementary processing element (PE) and a topology to connect functional groups of processing elements.

The Processing Element. A single processing element (PE), which is dedicated to process each pixel of a block, is depicted in Figure 2. The same processing element is

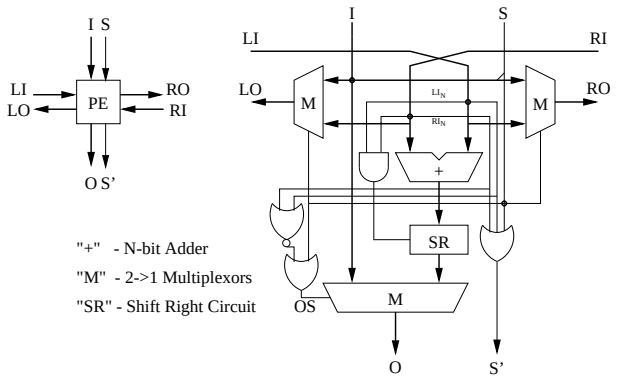


Figure 2: A Padding Processing Element

used for luminance and chrominance padding. The following equations describe the functionality of the processing element:

$$OS = (S \vee \overline{LI_N \vee RI_N}) \tag{1}$$

$$O = OS \cdot I \vee \overline{OS} \cdot [(LI_{\overline{N}} + RI_{\overline{N}}) >> i], \tag{2}$$

$$LO = S \cdot |\overline{S,I}| \vee \overline{S} \cdot RI, |RO| = S \cdot |\overline{S,I}| \vee \overline{S} \cdot LI, \tag{3}$$

$$S' = S \vee LI_N \vee RI_N; \tag{4}$$

where $i = LI_N \wedge RI_N$,
OS stands for Output Select signal,
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 (least-significant) bits of LI (bits 0 to N-1),
LI_N represents the N^{th} (the most-significant) bit of LI,
and $\overline{|S,I|}$ denotes the concatenation of bit S and vector I.

The operation of the PE is explained by the following:

- 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.

The Systolic Structure. To process a line from a macroblock, we implement the systolic structure of processing elements, depicted in Figure 3. 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 3. For example, if we implement eight such structures, we will be able to process eight lines in parallel. This is possible, because in the padding algorithm there is no data dependency between any two lines or columns. The data dependency is just between the pixels in the same line/column. Even a larger, two dimensional structure for processing a whole block in parallel is implementable. Implementations, which process more than one macroblock lines in a time, however, require higher data throughput and the utilization of more complicated addressing approaches would become necessary.

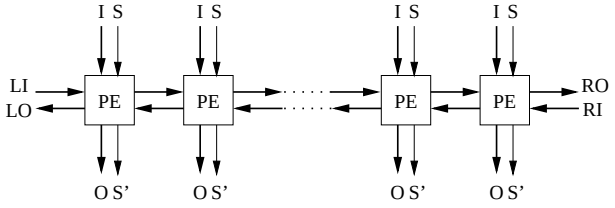


Figure 3: Single Scan Line/Column Padding Structure

We can easily evaluate the processing speed of the structure, given its operating frequency [1]. Let's assume a chain of n PE like the one, depicted in Figure 3, operating at frequency F_n [Hz]. The values of n with practical significance are 4, 8, 16. We state two more parameters of the particular implementation:
$\mathbf{N_n^{P8}}$ *and* $\mathbf{N_n^{P16}}$ *denote the numbers of cycles, necessary to process an 8-pixel (chrominance) and a 16-pixel (luminance) line respectively.* Some potential values of these parameters are shown in Table 4.
N_n^{P8} is a variable for $n < 8$ and a constant for $n \geq 8$. Identically N_n^{P16} is a variable for $n < 16$ and a constant for $n \geq 16$. This is due to the pixel-data dependency in a line, imposed by the padding algorithm, and the necessity of extra (data - dependent) numbers of cycles for structures with less than 16PE. Thus the processing of 16 pixels by any nPE configuration will take $\frac{N_n^{P16}}{F_n}$ [seconds] and for a 256-pixel luminance block- $\frac{16 \cdot N_n^{P16}}{F_n}$ [s]. Identically, the processing of two 8x8-pixel chrominance blocks will take $\frac{16 \cdot N_n^{P8}}{F_n}$ [s] of the same unit configuration. Since a macroblock consists of 256 luminance and 128 (2 x 64) chrominance pixels, padded vertically and horizontally, a whole macroblock will be padded for $\frac{32}{F_n} \cdot (N_n^{P8} + N_n^{P16})$ [s]. If we implement a configuration, which processes two and more (say k) 16-pixel lines in a time, we can formulate the *Processing speed* as follows:

$$Processing\_speed = \frac{F_n \cdot k}{32 \cdot (N_n^{P8} + N_n^{P16})} \qquad (5)$$

Formulation (5) is still valid for $n < 16$, assuming in this case that k=1.

Possible Configurations. For the line/column repetitive padding unit, there could be several configuration options. Some of them are:

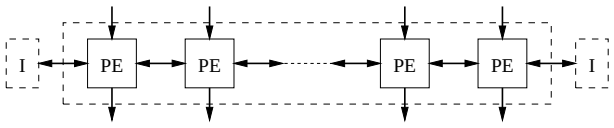
1. 16PE unit - processes one luminance line/column and two chrominance per operating cycle.
2. 8PE or 4PE unit - processes a half/quarter of luminance and one/half chrominance line/column. An additional control circuit is required, to maintain intermediate computational results.
3. 32, 64, ..., 256PE unit- processing more than two luminance and more than four chrominance lines/columns per operating cycle. The extreme configuration would process the whole macroblock.

Figure 4 depicts a general view of the discussed possible configurations of the padding unit. The blocks, named "I" are buffers which store the Initialization values for configurations, containing multiple of 16 PE. For Configurations with less than 16 PE, these buffers are used for storing intermediate values, in order not to cut the data propagation chain for longer (up to 16 pixel) lines.

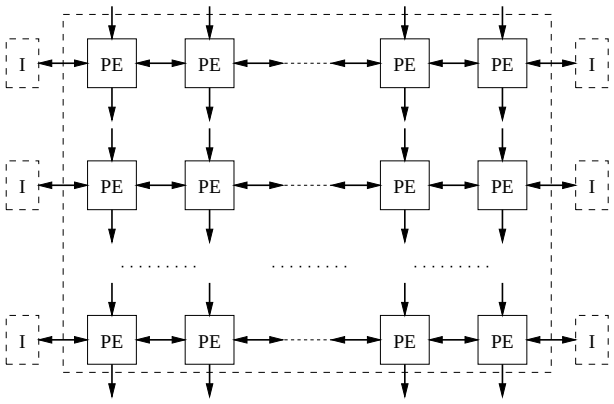
4. SIMULATION RESULTS AND EVALUATION

To evaluate the proposed structure, we have explored configurations with different numbers of PEs. The evaluation has been made in terms of *chip area* and *speed*. We have written synthesizable VHDL models of a single PE and a generic multi-element structure of PEs. To get realistic values for the parameters of the unit, we have synthesized these

[1]In this paper we distinguish (data) processing speed, measured in [macroblocks/sec] (or [MB/s]) from the device operating speed (frequency), measured in [Hz].



(a) Up to 16 PE



(b) 32, 64,..., 256 PE

Figure 4: Possible Configurations - *"I" denotes Initialization and/or Intermediate Result Buffer*

VHDL models for two popular FPGA families - Altera and Xilinx, using the standard synthesis and simulation tools, provided by the companies. 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 (hence cheaper) generations of FPGAs. We evaluated both the Xilinx xc4085xlpg559-09 [16] and the Altera epf10k20rc240-4 [1] chips, because they can be run at comparable frequencies (around 100 MHz). Their chip organization, however, is quite different and since area is estimated in different units for each of them, an area comparison between the mappings onto the two different chip families has not been done.

For both chip families we have evaluated structures of 4, 8 and 16 PEs and speed has been reported in MHz. Two extra evaluations for 32 and 64 PEs Xilinx mappings, have been done to illustrate how the data organization and the number of processing elements influence the performance of the unit.

Area and Speed Evaluation. Table 2 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 3 in similar manner but the units for the absolute area are referred to as LCs (Logical Cells).

The speed estimations for both FPGA families suggest similar results. Besides the operating frequency, measured in MHz, we also evaluated the actual data processing speed of the different configurations. 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* (Table 1).

Table 2: Results for the Xilinx xc4085xlpg559-09 chip

# PE	# CLB's		Speed		
	total	%	MHz	MB/s	
				Probable	Worst C.
4	45 of 3136	4	24.5	95 700	76 600
8	206 of 3136	7	18.2	162 500	142 200
16	419 of 3136	14	11.4	237 500	237 500
32	838 of 3136	27	**11.4**	**475 000**	**475 000**
64	1676 of 3136	53	**11.4**	**950 000**	**950 000**

Table 3: Results for the Altera epf10k20rc240-4 chip

# PE	# LC's		Speed		
	total	%	MHz	MB/s	
				Probable	Worst C.
4	254 of 1152	22	24.8	96 900	77 500
8	511 of 1152	44	19.8	176 800	154 700
16	1024 of 1152	88	13.4	279 200	279 200

For consistency with this definition, in the last two columns of Tables 2 and 3, we have estimated the processing speed in macroblocks per second ([MB/s]) according to (5) in its most probable and worst cases. The reported numbers indicate that the padding structure can meet the real-time requirements for a broad range of visual resolutions. If we consider the implementation with 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 279 167 MB/s, depending on the FPGA family.

Table 4: Values of N_n^{P8} and N_n^{P16}

n	Probable		Worst Case	
	N_n^{P8}	N_n^{P16}	N_n^{P8}	N_n^{P16}
4	2.5	5.5	3	7
8	1	2.5	1	3
16	0.5	1	0.5	1
16·k	0.5	1	0.5	1

The *Core* and *Main* profile levels of MPEG-4 require processing speeds in the range between 2970 and 244800 Boundary MB/s to maintain from 4 up to 32 VOPs. It is obvious that the operating speed ranges, achievable by the proposed padding unit, completely match the required values. Even the most demanding profile level, *level 4* of the *Main* MPEG-4 profile, requires 244 800 Boundary MB/s for a high resolution session type (1920 x 1088) and 32 objects. This rate is in the order of the reported speed results for the feasible padding unit implementations. Furthermore, the unit can also meet the requirements for the maximum operating speeds from Table 1 (5940 MB/s - 489600 MB/s). This allows the boundary macroblock detection to be avoided as a processing stage, preceding the repetitive padding and all blocks in the visual scene to be padded. The potentials of the structure indicate capabilities to meet even more-demanding future profiles of the visual data compression standards.

Data-Area-Speed Dependency. Results in Table 2 and Table 3 also indicate that the actual operating speed depends both on the number of processing elements and the

structure of the processed data. For configurations with less than 16 PE, the area-speed relation is not proportional. The situation is different for structures with more than 16 PEs. The last two rows of Table 2 illustrate the influence of the data organization (16x16 pixel macroblocks) and the configuration of the structure on then processing speed (for numbers of elements lower and higher than 16). Since any two different lines (columns) within a macroblock are padded independently from each other, by parallelizing the processing per lines (columns), we significantly increase the processing speed without changing the operating frequency. On the other hand, structures with less than 16 PE, require extra circuitry to maintain the intermediate computational results. Therefore the number of the processing cycles is much higher and depends on the data. Figure 5 depicts the discussed influence of the data organization on the area-speed relation. It is evident that for structures with more than 16 PE the speed increases with the same rate as the area does. These properties of the implementation should be taken into account when either the area or the speed constraints of the unit are crucial.

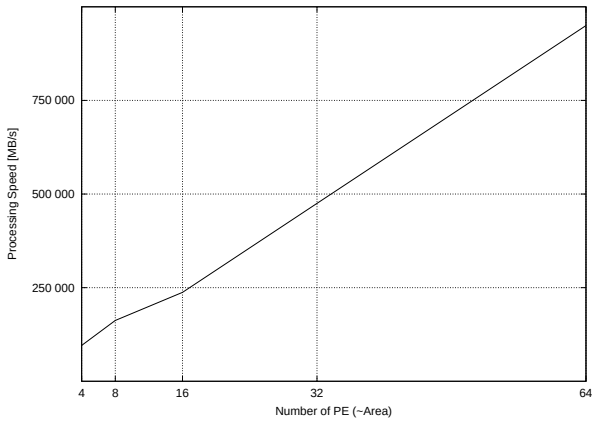


Figure 5: Data Structure Influence on the Performance *(Unit Mappings on Xilinx FPGA)*

5. CONCLUSIONS

In this paper we proposed a scalable padding unit, capable to process macroblocks in MPEG-4. The unit has been modeled with 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 trivial hardware costs. The significantly low operating frequencies of the proposed reconfigurable circuit make it an alternative of the unrealistic multi-thousand MIPS requirements of its software implementation. An operating frequency of up to 13.4 MHz allowed a processing speed of up to 279 200 MB/s to be achieved by only 16 PEs, mapped on relatively small Altera FPGA. Larger configurations of the unit were mapped on a yet cheap Xilinx FPGA and identical performance results were reported. Evaluations indicate that for configurations of more than 16 PE, the speed-area dependency is a linear function, unlike structures with less than 16 PE. With a processing speed of approximately 950 000 MB/s, a 64PE padding unit can achieve performance well above the required by the highest level of the most-demanding MPEG-4 visual profile.

6. 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.

7. 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] E. A. Edirisinghe, J. Jiang, and C. Grecos. Shape adaptive padding for MPEG-4. *IEEE Transactions on Consumer Electronics*, 46(3):514–520, August 2000.

[5] C. Heer and K. Migge. VLSI hardware accelerator for the MPEG-4 padding algorithm. In *IS&T:SPIE Conference on media processors*, volume 3655, pages 113–119, 1999.

[6] ISO/IEC JTC11/SC29/WG11 N2802. ISO/IEC 14496-2. Generic Coding of Audio-visual Objects- Part2: Visual. Final Proposed Draft, July 1999.

[7] ISO/IEC JTC11/SC29/WG11, N3312. MPEG-4 video verification model version 16.0.

[8] ISO/IEC JTC11/SC29/WG11 N4030. MPEG-4 overview, March 2001.

[9] A. Kaup. Object-based texture coding of moving video in MPEG-4. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(1):5–15, February 1999.

[10] J. Kneip, S. Bauer, J. Vollmer, B. Schmale, P. Kuhn, and M. Reissmann. The MPEG-4 video coding standard - a VLSI point of view. In *IEEE Workshop on Signal Processing Systems,(SIPS98)*, pages 43–52, 8-10 Oct. 1998.

[11] J.-H. Moon, J.-H. Kweon, and H.-K. Kim. Boundary block-merging (BBM) technique for efficient texture coding of arbitrarily shaped object. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(1):35–43, February 1999.

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

[13] 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.

[14] 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.

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

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