Building Blocks for MPEG Stream Processing

Stephan Wong, Jack Kester, Michiel Konstapel, Ricardo Serra, and Otto Visser
Computer Engineering Laboratory,
Electrical Engineering Department,
Delft University of Technology,
Stephan@Dutepp0.ET.TUDelft.NL

Abstract— **In the MPEG-1 multimedia standard, no encoding scheme is defined, but a generally accepted scheme includes the following building blocks: discrete cosine transform, quantization, zig-zag scanning, run-level coding, variable-length encoding, and motion estimation. In this paper, we describe the investigation into how such building blocks can be implemented in reconfigurable hardware, more specifically field-programmable gate arrays (FPGAs). The investigation focuses on the area requirements and the expected speed of such building blocks. Since our investigation will be applied to increase the performance of a programmable processor (running an MPEG-1 software application) augmented with an FPGA structure, we have selected four known to be time-consuming operations for implementation in FPGA: forward discrete cosine transform, the quantization, the Huffman encoding, and the sum of absolute differences. All designs were implemented by writing high-level VHDL code and target two FPGA families from Altera Corp.: FLEX10K and APEX20K. The synthesis results show that the implementations can be clocked between 36 and 162 MHz and that the area utilization is small compared to the largest available FPGA chip within the APEX20K family.**

Keywords— **field-programmable gate array, forward discrete cosine transform, quantization, Huffman encoding, sum of absolute differences**

I. Introduction

In digital video coding, a multitude of compression techniques are employed in order to reduce the size of the digital representation of the video data. These techniques exploit redundancies found within the digital representation by removing the information that can be reconstructed (up to a certain degree) from the remaining information or that cannot be observed by the human eye. In order to avoid competing (proprietary) industry standards, several multimedia standards for audiovisual content (e.g., MPEG-1, MPEG-2, and MPEG-4) were defined that specify the utilization of specific compression techniques. These techniques are tailored towards the intended application, e.g., television broadcast over satellites or video streaming over the Internet. In this paper, we discuss several compression techniques specified in the MPEG-1 standard and then focus on specific operations that are required to perform these techniques. The multimedia operations we focus on are: forward discrete cosine transform, the quantization, the Huffman encoding, and the sum of absolute differences. These operations can be implemented in hardware or software or both. In this paper, we additionally focus on the implementation of these operations in reconfigurable hardware, like field-programmable gate arrays (FPGAs). The reasons to utilize FPGAs are discussed in the following.

Traditionally, the design of embedded multimedia processors were very much similar to the design of microcontrollers. This meant that for each targeted set of multimedia applications, an embedded multimedia processor needed to be designed in specialized hardware (commonly referred to as Application Specific Integrated Circuits (ASICs)). In the early nineties, we were witnessing a shift in the embedded processor design approach fuelled by the need for faster time-to-market times. In embedded processor design, this resulted in the utilization of programmable processor cores augmented with specialized hardware units implemented in ASICs. Consequently, time-critical tasks were implemented in specialized hardware units while other tasks were implemented in software to be run on the programmable processor core [8]. This approach allowed a programmable processor core to be reused for different sets of applications and only the augmented units need to be (re-)designed for specific application areas.

Currently, we are witnessing a new trend in embedded processor design that is again quickly reshaping the embedded processor design. Instead of implementing the time-critical tasks in ASICs, these tasks are to be implemented in field-programmable gate arrays (FPGA) structures or comparative technologies [4], [9], [11], [5]. The reasons for and the benefits of such an approach include the following:

- **Increased flexibility:** The functionality of the embedded processor can be quickly changed without requiring another roll-out of the embedded processor itself and design faults can be quickly rectified. It also allows for quick adaptation of new (possibly unforeseen) developments.

• **Sufficient performance:** The performance of FPGAs has increased tremendously and is quickly approaching that of ASICs [2]. This seems to be mainly due to the faster adaptation of new technological advancements by FPGAs than by ASICs.

• **Faster design times:** Faster design times are achieved by re-using intellectual property (IP) cores or by slightly modifying them. More importantly, high-level design languages (such as VHDL) can be used in the design process and thereby speeding it up significantly.

The mentioned advantages and enabling FPGA have even resulted in that programmable processor cores are implemented on the same FPGA structure, e.g., Nios from Altera [1] and MicroBlaze from Xilinx [3].

In this paper, we investigate the area requirements and expected speed of implementing four distinct multimedia operations as four distinct implementations in FPGA hardware, namely forward discrete cosine transform, quantization, Huffman encoding, and sum of absolute differences. The utilization of four distinct implementations instead of an integrated solution is derived from the fact that we intend to augment a single (most likely small) FPGA structure to a programmable processor core. The implementations are intended to increase the performance of an MPEG-1 software application running on the processor core. Therefore, we opted to perform the hardware design separately for each multimedia operation. In this way, each design can be perceived as a single accelerator that can be used improve the performance of the processor core. Furthermore, the designs described in this paper are such that allow them to be synthesized in order to obtain area and performance estimates with little attention paid to their compatibility. In addition, we have to note that only small design changes are needed to connect the four implementations when a complete MPEG-1 solution is sought after. Finally, we target two FPGA families from Altera Corp.: FLEX10K and APEX20K.

This paper is organized as follows. In Section II, we describe in more detail a generally accepted video encoding scheme of MPEG-1 and introduce the four targeted multimedia operations. In Section III, we discuss the implementation details of the four multimedia operations. In Section IV, we present the synthesis results regarding area and clock speed. In Section V, we conclude this paper with some concluding remarks.

II. VIDEO ENCODING

The MPEG-1 multimedia standard does not specify a strict encoding scheme since it would limit its implementation possibilities. This less stringent requirement allows the industry and the academic world to investigate a wide variety of encoding schemes with different design parameters, e.g., performance, speed, cost-effectiveness, etc.. However, from these investigations it is possible to deduce a general encoding scheme on which many implementations are based. The general MPEG-1 video encoding scheme is depicted in Figure 1.

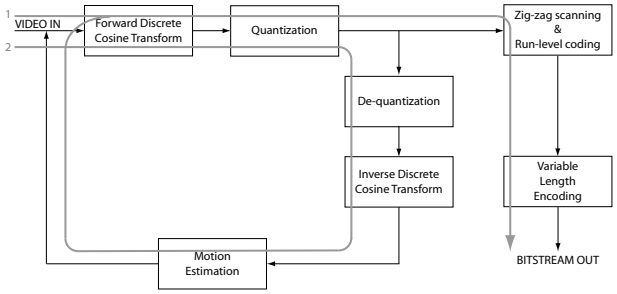

Fig. 1. MPEG-1 encoding scheme.

In this scheme, two distinct encoding paths can be observed. The first path runs straight from 'VIDEO IN' to 'BITSTREAM OUT' and is mainly concerned with the coding of the video frames as if they were single pictures. More specifically, only redundant information found within one single video frame is utilized in order to achieve compression. The *forward discrete cosine transform* (FDCT) utilizes space-to-frequency transformations to achieve compression, i.e., pel[1] values (in the space domain and arranged in a 2-dimensional (2D) regular array) denoted by $f(x, y)$ are transformed into $F(\mu, \nu)$ in the frequency domain given by the following equation:

$$F(\mu,\nu) = \frac{C(\mu)}{2} \times \frac{C(\nu)}{2} \times \sum_{y=0}^{7}\sum_{x=0}^{7} f(x,y) \cdot cos[(2x+1)\mu\pi/16] \cdot cos[(2y+1)\nu\pi/16] \quad (1)$$

where μ and ν are the horizontal and vertical frequency indices, respectively, and the constants, $C(\mu)$ and $C(\nu)$, are given by:

$$C(\mu) = \tfrac{1}{\sqrt{2}} \text{ if } \mu = 0$$
$$C(\mu) = 1 \text{ if } \mu > 0$$

Without delving into the theoretical background of the FDCT, we can state that the FDCT transforms the input values (exhibiting redundancy) into a few uncorrelated transform coefficients (also arranged in a 2D array). The main characteristic of this transform is that the information represented by the (redundant) input values is being

[1] Pel stands for picture element and represents the smallest color data unit of a picture or video frame.

mainly compacted into the low-frequency transform coefficients. More specifically, the low-frequency transform coefficients have high values while high-frequency ones have low values or are mostly zero. Therefore, by coding only a smaller number of transform coefficients instead of the whole array of the input values, compression is achieved. In addition, the remaining transform coefficients do not need to be represented using full accuracy and thus enabling the next stage to achieve higher compression. The *quantization* is utilized to reduce the accuracy of the transform coefficient's representation through division of the transform coefficients by pre-determined quantization factors. In most cases, this results in that most of the remaining non-zero transform coefficients become zeroes. The *zig-zag scanning* converts (in a pre-determined manner) the 2D array of quantized transform coefficients into a 1D array. The main purpose is to serialize the quantized transform coefficients such that (preferably long) sequences of zeroes can be identified. The *run-level coding* scans the 1D array from beginning till end and produces run-level pairs in the following manner. First, it is determined how many zeroes precede a non-zero value and this is called the run (a zero run is possible). Then, the level which is the non-zero value terminating the sequence of zeroes (if any) is noted. When put together, a run-level pair is generated. In the case, that no non-zero value is present till the end of the 1D array, a special symbol called end-of-block (EOB) is generated. In the *variable length encoding*, predetermined Huffman tables are utilized to assign variable bit sequences to such run-level pairs (or the EOB) in order to generate the final bitstream. In these tables, shorted bit sequences are assigned to frequently occurring run-level pairs and thereby reducing the average size of the resulting bitstream.

The second path in Figure 1 exploits additional information from reference frames in order to exploit similarities between these frames and the current to be encoded frame. The reference frames can be either preceding or following the current frame. Due to the fact that two temporally close frames exhibit a considerable amount of similarities, the encoding of their differences requires significantly less storage and thereby achieving compression. As reference frames, the decoded versions of previously encoded frames are used due to the utilization of lossy compression techniques such as quantization. As a result, the encoded differences are more closely matched with the decoded frames in the decoder. To this end, the *inverse discrete cosine transform* and the *de-quantization* are being utilized. However, a well-known issue that prohibits a straightforward implementation relates to motion found in video scenes, i.e., the similarities found between frames are not always located at the same position. This has led to the introduction of *motion estimation* which attempts to capture the 'movement' of a macroblock (a 16×16 array of pels) in the reference frame in relation to the to be encoded macroblock in the current frame. This is done by searching an area in the reference frame for the 'best' match according to a specific metric. Two widely used metrics are the mean square error (MSE) and the mean absolute difference (MAD) of which the latter one is more commonly used since it is computationally less intensive to calculate. The MAD is given below:

$$MAD(x,y,r,s) = \frac{1}{256}\sum_{i=0}^{15}\sum_{j=0}^{15}|(A_{(x+i,y+j)} - B_{((x+r)+i,(y+s)+j)})| \quad (2)$$

with $0 \leq x$,$y <$ framesize
with (r,s) being the motion vector
with $A_{(x,y)}$ being a current frame pel at (x,y)
with $B_{(x,y)}$ being a reference frame pel at (x,y)

Since the division by 256 can be performed by a simple shift operation in computer arithmetic, we focus on the sum of absolute differences(SAD). The relation between the SAD and the MAD is given below:

$$MAD(x,y,r,s) = \frac{SAD(x,y,r,s)}{256} \quad (3)$$

III. Four Multimedia Operations Design

In the previous section, we have highlighted a generally accepted video encoding scheme for the MPEG-1 multimedia standard. Within this scheme, we can identify four multimedia operations that are compute-intensive and therefore are prime targets to be implemented in specialized hardware. More specifically, we target field-programmable gate arrays (FPGAs). The four operations are: forward discrete cosine transform, quantization, sum of absolute differences, and Huffman encoding. In this section, we briefly discuss the four multimedia operations and some specifics of the designs. We have to note that the designs are distinct and can not be connected to each other without small design changes. The reason for this apparent incompatibility is due to the fact that we do not envision large-scale FGPA structures to be augmented to programmable processor cores in embedded systems design. Therefore, we opted to perceive the four designs as separate accelerators intended to improve the performance of an MPEG-1 software application running on a programmable processor core.

Forward discrete cosine transform (FDCT): The 8×8 2D FDCT given in Equation 1 can be split into 16 1D FDCTs by first applying the 1D FDCTs over the 8 rows and then over the resulting 8 columns (or the other way around). In order to save area, we have opted to implement the 1D FDCT described in [6] which utilizes 11 multiplications and 29 additions. The input of the FDCT is 9 bits and the output is 12 bits. Furthermore, our synchronized design is completely pipelined and it takes 5 clock cycles to generate the first result.

Quantization: The quantization step in our design has been combined with the zig-zag scanning. The resulting quantization unit is depicted in Figure 2.

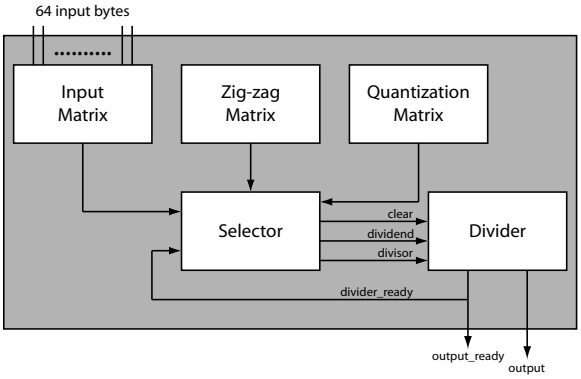

Fig. 2. Quantization combined with zig-zag scanning.

Figure 2 depicts the internal organization of the quantization unit. The input matrix stores the input values that are generated after the FDCT. We have opted to utilize 8 bits per input value[2] in order to diminish the pin requirements by $64 \cdot 4 = 256$ pins. The zig-zag matrix stores the order in which the zig-zag scanning must be performed. The quantization matrix stores the quantization factors that are utilizes as divisors. The selector unit selects the correct quantization factor for each input value depending on the zig-zag scanning order. Subsequently, both the input value (dividend) and quantization factor (divisor) are forwarded to the actual divider unit. The output of the quantization unit is again an 8-bit value. Finally, it takes 12 cycles to produce a quantized DCT coefficient.

Huffman coding: The Huffman coding design translates each run-level pair or end-of-block (EOB) symbol to a bit sequence according to a Huffman table[7] (pre-defined in the MPEG-1 standard). When an undefined run-level pair has been encountered, a special escape code must be generated. The input is 6-bit value specifying the run and an 8-bit value specifying the level. The output bit length is variable, but never exceeds 20 bits. Therefore, the output of the Huffman encoding is placed in an 20-bits output buffer accompanied with a masking buffer that specifies which bit position in the output buffer contains a valid output bit. Finally, the design is pipelined and produces a result 2 clock cycles after the input has been asserted.

Sum of absolute differences (SAD): In our investigation, we have opted to implemented the SAD design based on the one described in [10]. As indicated by Equations 2 and 3, the SAD operation is performed on macroblock A (16×16 array of pels) in the current frame and macroblock B in the reference frame. First, the absolute difference must be calculated and then these are all added together to produce the final result. The innovation described in [10] lies in the fact that the calculation of the absolute difference is translated into an addition that can be merged into the adder tree. The internal organization of the implemented SAD design is depicted in Figure 3.

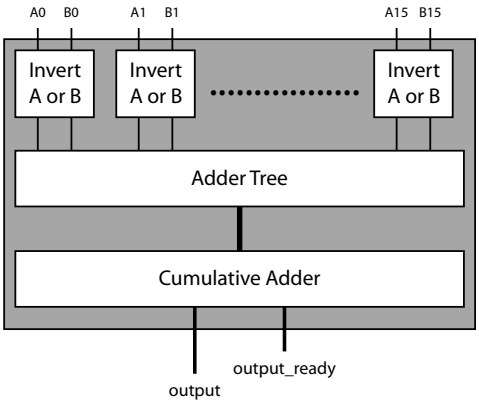

Fig. 3. The sum of absolute differences unit.

In Figure 3, a SAD unit is depicted that calculates the 16×16 SAD operation by iteratively processing the rows. First, the 'Invert A or B'-block determines whether to invert input A or input B. Afterwards, the inverted value and non-inverted value are passed on to the adder tree. Then, the adder tree calculates the intermediate sum of each row which. Finally, all the intermediate sums are added by the cumulative adder one by one as they become available. The input of the SAD unit consists of 32 8-bit values and the output consists of a 16-bit result and the 'output_ready' bit signalling a valid result. It takes 27 cycles to complete the first result and since the 16 row calculations are performed serially, the subsequent results is produced every 16 cycles.

IV. Synthesis Results

The four designs discussed in the previous section have been implemented by writing high-level VHDL code. Then, the functionality of the VHDL code is verified

[2]This is different from the 12 bits output generated from the FDCT design, but as already mentioned all designs are distinct from each other.

	DCT		Quantization		Huffman		SAD	
FLEX10K								
max. clock speed	58	MHz	44	Mhz	36	MHz	49	MHz
area	954	LCs	471	LCs	540	LCs	2772	LCs
APEX20K								
max. clock speed	106	MHz	135	MHz	52	MHz	162	MHz
area	1000	LCs	470	LCs	485	LCs	2126	LCs

TABLE I
SYNTHESIS RESULTS.

by utilizing the MAX+plus II (Baseline v 10.1) software package from Altera Corp. Finally, the VHDL code is synthesized by utilizing the LeonardoSpectrum software package from Exemplar Logic Inc. and targets the FLEX10K and APEX20K FPGA families from Altera Corp. The synthesis results are presented in Table I. In this table, we can observe that the APEX20K family 2-3 times faster than the FLEX10K family for the same implementation. Furthermore, the results show that the area utilization (expressed in logic cells (LCs)) are similar in both families. When a complete and stand-alone MPEG-1 solution is sought after, i.e., not in conjunction with a programmable processor core, the four mentioned designs can be easily connected to each other with minimal design changes. For example, the bit length of the inputs to the quantization need to be extended from 8 bits to 12 bits. Considering that the largest chip to date of the ACEX20K family contains > 51.000 logic cells, several design could be parallelized in order to increase the performance by exploiting the additional area.

V. CONCLUSION

In this paper, we described our investigation in implementing specific multimedia operations (found in video coding) on field-programmable gate arrays (FPGAs). The investigation focused on the area requirements and expected speed of such building blocks. Since our investigation is placed in the framework of programmable processor cores augmented with FPGA hardware (units), we have selected four known to be time-consuming multimedia operations to be implemented: forward discrete cosine transform, quantization, Huffman encoding, and sum of absolute differences. The resulting designs are such that they are intended to be utilized separately from each other as they are intended to improve only a specific part of the software MPEG-1 application. All the designs were implemented by writing high-level VHDL code and synthesized afterwards by targeting two FPGA families from Altera Corp.: FLEX10K and APEX20K. The synthesis results show that the implementations can be clocked between 36 and 162 MHz and that the area utilization is small compared to the largest available FPGA chip within the APEX20K family. The results presented in this paper can be utilized in a cycle-accurate simulator in order to gain insight into the potential performance increase when compared to a software-only implementation of the MPEG-1 encoding scheme.

REFERENCES

[1] Nios Embedded Processor. http://www.altera.com/products/devices/excalibur/exc-nios_index.html.

[2] Virtex-II 1.5V FPGA Family: Detailed Functional Description . http://www.xilinx.com/partinfo/databook.htm.

[3] Xilinx MicroBlaze. http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=microblaze.

[4] D. Cronquist, P. Franklin, C. Fisher, M. Figueroa, and C. Ebeling. Architecture Design of Reconfigurable Pipelined Datapaths. In *Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI*, pages 23–40, March 1999.

[5] J. Hauser and J. Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *Proceedings of the IEEE Symposium of Field-Programmable Custom Computing Machines*, pages 24–33, April 1997.

[6] C. Loeffler, A. Ligtenberg, and G. Moschytz. Practical Fast 1-D DCT Algorithms With 11 Multiplications. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 988–991, 1989.

[7] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall. *MPEG Video Compression Standard*. Digital Multimedia Standard Series. Chapman and Hall, 1996.

[8] S. Rathnam and G. Slavenburg. An Architectural Overview of the Programmable Multimedia Processor, TM-1. In *Proceedings of the COMPCON '96*, pages 319–326, 1996.

[9] R. Razdan and M. Smith. A High-Performance Microarchitecture with hardware-programmable Functional Units. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 172–180, November 1994.

[10] S. Vassiliadis, E. Hakkennes, S. Wong, and G. Pechanek. The Sum-Absolute-Difference Motion Estimation Accelerator. In *Proceedings of the 24^{th} Euromicro Conference*, 2000.

[11] R. Wittig and P. Chow. OneChip: An FPGA Processor with Reconfigurable Logic. In *Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 126–135, April 1996.