Mapping of Motion Estimation on a VLIW Processor Template

Antanas Snirpunas†, Antoine van Wel‡, and Stephan Wong†

†Computer Engineering Laboratory,
Electrical Engineering Department
Delft University of Technology,
Delft, The Netherlands
santana@elen.ktu.lt,
stephan@Dutepp0.ET.TUDelft.NL
http://ce.et.tudelft.nl

‡Philips Research Labs
Eindhoven, The Netherlands
Antoine.van.Wel@philips.com

Abstract— **In designing video encoding systems, several issues exist such as real-time constraints, low power, area, and flexibility that must be addressed when designing such systems. While these issues can be addressed by different solutions, the challenge lies in finding a single solution that provides an adequate balance between the mentioned issues. In this paper, we describe a VLIW (Very Long Instruction Word) processor that is based on a scalable template and that addresses the above-mentioned issues. We optimize the VLIW processor by incorporating new functional units and extending existing ones in order to support motion estimation. Results show that our solution in supporting the full-search algorithm can perform real-time video encoding. The estimated area of the VLIW is 1.1mm^2. Furthermore, our solution is adaptable in the sense that other less compute-intensive search algorithms can also be supported. The presented work is an initial step towards a fully programmable video encoding processor that is also able to support operations such as DCT (Discrete Cosine Transform), IDCT (Inverse Discrete Cosine Transform), quantization, etc.**

Keywords— **Video encoding, motion estimation, VLIW, scalable VLIW template.**

I. Introduction

In recent years, we are witnessing a shift of consumer video applications into the domain of mobile handheld mass-produced devices [13]. At the same time, there exists a steadily growing demand for high-performance hardware for video applications. Therefore, the design and implementation goals of video application-related systems should include the following: achieving sufficiently high performance with relatively low power consumption and at relatively low cost. Achieving high performance (in other words, satisfying real-time constraints) and low power consumption at the same time appear to be contradictory. However, it can be achieved by optimizing algorithms such that computationally intensive calculations are mapped on specialized hardware that exploits potential instruction and data parallelisms. Cost may be decreased by reducing the area of hardware and by utilizing application-specific flexible hardware, which allows modification after it is manufactured and which can be reutilized in subsequent design cycles. The design and implementation goals of video processing applications can be achieved by identifying the most computationally intensive operations in video processing.

Video encoding is usually an initial stage in many video processing applications found in consumer electronics, such as digital video cameras, etc.. Within video encoding, motion estimation plays an important role as it is one of the most computationally intensive operation [5], [6]. In addition, higher performance of video encoding can be achieved by optimizing the motion estimation algorithm and its hardware.

The goal of our project is to implement motion estimation, as part of a video encoder, on a VLIW template-based architecture. A VLIW processor allows us to utilize a number of functional units in parallel, each dedicated to a specific instruction set. High performance of a VLIW can be achieved by utilizing a number of functional units in parallel [1]. Video data in combination with an algorithm for the motion estimation process allows us to utilize instruction and data-level parallelisms in order to achieve high performance. This is accomplished by mapping computationally intensive instructions on a number of functional units that operate concurrently. Moreover, the performance can be increased by extending the instruction set of a standard functional unit. Thereby, we can support a non-standard operation that takes more than one cycle to execute on a standard functional unit.

In this paper, we present the implementation of a VLIW processor that is based on a scalable template. The implementation is optimized by incorporating new functional units and extending existing ones in order to support mo-

tion estimation. Results show that our solution can support the full-search algorithm, in the real-time encoding of QCIF frames (176×144) at 30 frames per second with a search depth of 6 pels. The estimated area of the VLIW is 1.1mm^2.

The remainder of this paper is organized as follows. In Section II, we present the background on video encoding and the utilized VLIW processor template. In Section III, we discuss our implementation on the mentioned VLIW processor template. In Section IV, discuss how our design has been verified and we present an evaluation in terms of performance and area. In Section V, we present some concluding remarks.

II. Background

In this section, we present some background on both video encoding and the utilized VLIW processor template.

A. Video Encoding

Digital video data (further, video data) in its raw format is enormous in size and therefore, it is expensive to store it, process it, or transmit it. As a result, video data is being encoded to reduce its size in order to satisfy storage, processing and transmission requirements [4]. High compression ratios of video data can be achieved by exploiting the high degree of data redundancy in video sequences [8]. There are three types of redundancy that are exploited for video encoding [9]:

- **Spectral redundancy** is exploited to compress video data by converting pel[1] values from in the RGB colorspace format[2] to the YC_rC_b colorspace format, where each single pel is encoded with one luminance (Y) and two chrominance (C_r, C_b) components.
- **Spatial redundancy** represents the correlation of pel values in a frame.
- **Temporal redundancy** is the redundancy found between video frames, which are usually similar unless when there is a scene change or when there are moving objects.

In this paper, we address the temporal redundancy, which is specific for encoding video data. To remove the temporal redundancy, a predictive coding technique can be employed [10]. This technique can be utilized to encode consecutive frames that usually do not differ very much, in order to achieve higher compression. In this case, pels in one frame can be predicted by pels at the same location in a consequent frame. However, such an approach is not very efficient to implement, because objects tend to move in consecutive frames. Motion estimation has been introduced as a technique to capture the motion of objects in a video sequence. **Motion estimation** facilitates the removal of temporal redundancies between consecutive frames by estimating the movements of objects in a video sequence [12]. The motion estimation can be performed for each pel in a frame (pel-based motion estimation) or for a block (block-based motion estimation). In the remainder of this paper, we focus solely on block-based motion estimation.

[1]Pel is the smallest element of any system that represents data in the form of two-dimensional arrays of visual information [11].

[2]Colorspace format is a method for representation of a single pel.

B. Block-based Motion Estimation

In block-based motion estimation, the current frame is divided into macroblocks (a 16×16 array of pels) and the relative location of the most closely resembling macroblocks in the reference frame must be determined (see Figure 1. Motion estimation is performed by comparing a certain macroblock in a current frame with the set of macroblocks in a reference frame (see Figure 1) in order to find the best match [7], or, in other words, the smallest difference between two macroblocks. In order to better understand motion estimation, we define the following:

- **Search area** (see Figure 1) is the part of a reference frame where candidate motion vectors are restricted.
- **Candidate motion vector** is the definition of spatial displacement of a reference macroblock from a position of current macroblock.
- **The best motion vector** is a candidate motion vector which points to the best matching reference macroblock.

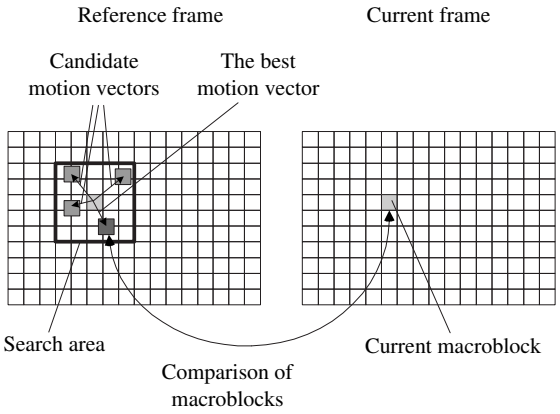

Fig. 1. Comparison between macroblocks in the current and in the reference frame.

The speed in finding the "best" matching motion vector is determined by the following: search area, search algorithm, and cost function. A large search area translates into many more comparisons to be made and thus decreasing the speed. An extensive search algorithm, i.e., one with many steps, also translates into a larger number of comparisons to be made. A complex cost function, which

determines what is the “best” match, also translates into a decrease of speed, because many more (or more complex) operation must be performed. On the hand, having a large search area, an extensive search algorithm, and a complex function might result in finding a much better “best” matching motion vector.

The choice of a search algorithm, as well as a cost function depends on the accuracy of the best motion vector needed and the speed and complexity requirements of a video encoder. Accuracy is needed to find a better match between two macroblocks in order to achieve higher compression ratio of video data.

C. VLIW Processor Template

The A|RT (Algorithm to Register Transfer) Designer Pro is a CAD (Computer Aided Design) tool, which is utilized to implement an algorithm in digital synchronous hardware [2]. More precisely, the tool helps to optimally translate an algorithm written in the C programming language into an RT (Register Transfer) level hardware description of a processor-like architecture. In order to efficiently implement an algorithm, the C code of the motion estimation process is described in such a way that A|RT Designer recognizes possible instruction-level parallelisms.

The A|RT Designer CAD tool offers a template-based VLIW architecture that is tuned to support the motion estimation process. With the template-based VLIW we can explore the balance between the performance of fixed-functionality specialized hardware (in other words, ASIC) and the flexibility of a general-purpose processor. Unlike general-purpose processors and ASICs, the flexibility and specialization of a template-based VLIW processor is not fixed. It is therefore possible to shift the implementation to either the ASIC side or to its general-purpose processors counterpart by introducing extra resources that can increase the flexibility [3].

The generated processor consists of functional units that are controlled by a VLIW-type controller. Such a structure is scalable for parallelism as well as for performance. The main difference between a general VLIW structure and the current template-based architecture lies in the type of register files utilized. For a general VLIW structure there is typically one register file utilized for all functional units, whereas in a VLIW structure, implemented by A|RT Designer, each functional unit can have its own register file thus making a VLIW implementation scalable.

A template-based VLIW consists of a set of functional units that operate in parallel (see Figure 2). A|RT Designer supports standard as well as custom designed functional units.

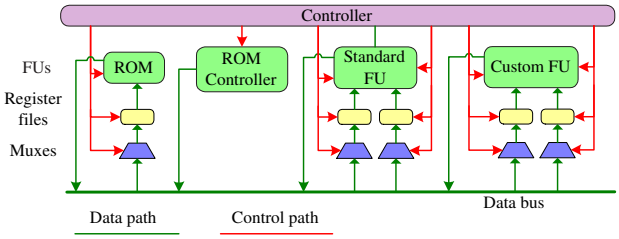

Fig. 2. Template-based VLIW.

III. Implementation

The main focus of implementing the motion estimation process on a VLIW processor template is on the design of specific functional units and on the extension of the instruction set that could accelerate the motion estimation in video encoding. Consequently, we identify the most time-critical functions, that have large influence on overall performance of the motion estimation process, map them on existing standard and new functional units.

For the implementation, the model of motion estimation unit has been defined in the C programming language. This model allows us to verify its functionality at software-level and to implement it on hardware with the A|RT Designer. The motion estimation model is purposely defined not to be dependent on a certain search algorithm[3] in order to design a more general structure of the VLIW processor. Therefore, the hardware/software partition splits the differences and commonalities of the operations in the motion estimation model. The different operations, such as the generation of a candidate motion vector set and refinement of a candidate motion vector, are specified at software level. Whereas, the common operations, such as evaluation of candidate motion vectors and fetching pel[4] data to an internal memory, are implemented at hardware level. Both operations require more attention due to the fact that they are the most time-critical operations of the motion estimation process implemented in hardware.

Both operations are refined with the help of A|RT Designer in order to find the most time-critical functions that if improved could accelerate the motion estimation process. The absolute difference operation is the most time-critical function for evaluation of candidate motion vectors. Whereas, data shuffling is the most time-critical function for fetching pel data to an internal memory. The absolute difference function is implemented as an extension to the instruction set of the standard ALU functional unit. Shuffling is implemented as a custom defined functional

[3]Here, the sum of absolute differences is utilized as a cost function.
[4]Only the luminance component is utilized for evaluation of candidate motion vectors.

unit, which, in addition, includes memory block for storing pel values of reference macroblocks and a current one.

During the instruction scheduling and resource allocation process in A|RT Designer, we are able to apply certain instructions to a particular functional units. Their size is reduced by optimizing the C description of the motion estimation model. Since video data in all units of an encoder, except the motion estimation, is being processed as a number of 8×8 pel-blocks we decided to implement the structure of the VLIW capable of processing 8-pels at a time. All pel-related functions, such as loading from internal memory, bilinear interpolation, absolute difference calculation etc., process 8 pels at a time. Such a structure of the VLIW still allows to flexibly implement other, video encoder-related, functions, such as DCT, quantization etc.

IV. Verification and Evaluation

Behavioral verification is performed by utilizing VHDL modeling and at netlist-level in CMOS12 technology with the Cadence NCsim tool. The Ambit CAD tool is utilized to synthesize the netlist-level description. The performance and hardware resources are evaluated according to the results of the A|RT Designer and NCsim simulations. Whereas area is estimated according to the results of Ambit synthesis. Verification is performed with the following specifications:

- Clock-speed is 100MHz.
- One frame, QCIF (176×144), is being processed.
- Macroblock's width is 16 pels.
- Internal memory size is 11×5 macroblocks.
- Full search algorithm is utilized.
- Search depth is 6 and motion vectors are represented in pel/quarter accuracy.

According to these specifications, the motion estimation process was performed correctly. The hardware-level simulation results match the ones acquired during software-level simulations.

A. Structure

Table I shows the set of functional units (together with their size) utilized in the implementation of the VLIW.

The number and the size of hardware resources (other than functional units), such as register files, multiplexors also have an impact on the performance, power consumption, and area figures of the VLIW. The size of these resources is defined by the A|RT Designer automatically, although, they might be directly controlled by modifying the C source code and by assigning instructions to the particular functional units. The size of a register file depends on the life-time of variables applied to a particular functional unit where a register file belongs to. The size of multiplexors depends on the number of data transitions among functional units.

Functional unit	Parameters		Quantity
	Input width	Output width	
ASB_MS (pel data interface)	2×32	32	1
ROM (Read Only Memory)	7	8	1
ROMCTRL (memory for constants)	-	15	1
ALU (Arithmetic Logic Unit)	2×32	32	3
ALU	2×16	16	6
ACU (Address Computation Unit)	3×16	16	2
ACU	3×32	32	1
MULT (Multiplication Unit)	2×16	32	1
MULT	2×9	18	8
IM_RB (Internal memory for reference macroblocks)	$16{,}2 \times 32$	8×8	1
IM_CB (Internal memory for a candidate macroblock)	$16{,}2 \times 32$	8×8	1

TABLE I
FUNCTIONAL UNITS UTILIZED IN THE IMPLEMENTATION OF THE VLIW PROCESSOR.

The width of a very long instruction word in the VLIW processor is a good parameter for performing optimizations during the design with A|RT Designer. The implemented VLIW processor has a very long instruction word which is 413 bits wide and the motion estimation process requires 128 such instructions in total to complete.

B. Performance Evaluation

According to the clock cycle count report, generated by A|RT Designer, we can estimate the number of clock cycles required to complete the motion estimation process for a one frame. This allows us to estimate the performance of the implemented VLIW processor. Although this report is generated when full search of motion estimation is employed with a certain search depth, it is relatively trivial to estimate the number of clock cycles required for other search algorithms and various frame sizes, because the generated VLIW processor is not dependent on a particu-

lar search algorithm. According to the accuracy of motion vectors, we can estimate the number of clock cycles for both cases: when motion vectors are represented in pel or quarter accuracy. The following equations 1 and 2 define the number of clock cycles to complete the motion estimation process when motion vectors are in pel and quarter accuracy accordingly.

$$\begin{aligned} C = SAW\_cols \cdot SAW\_rows \cdot (B\_width \cdot 7 + 3) + \\ + F\_rows \cdot F\_cols \cdot ((6 + B\_width \cdot 7) + \\ + CMV\_count \cdot (B\_width \cdot 5 + 21) + \\ + SAW\_rows \cdot (B\_width \cdot 7 + 3)) + 50 \end{aligned} \quad (1)$$

where:
C - the number of clock cycles required to complete motion estimation for one frame.
SAW_rows, SAW_cols - the number of rows[5] and columns, respectively, in the search area window that defines the size of the internal memory.
B_width - the width of a macroblock, which is specified as 16 pels in the design.
F_rows, F_cols - the number of rows and columns, respectively, in a frame.

$$\begin{aligned} C = SAW\_cols \cdot SAW\_rows \cdot (B\_width \cdot 7 + 3) + \\ + F\_rows \cdot F\_cols \cdot ((6 + B\_width \cdot 7) + \\ + CMV\_count \cdot (B\_width \cdot 14 + 36) + \\ + SAW\_rows \cdot (B\_width \cdot 7 + 3)) + 50 \end{aligned} \quad (2)$$

According to the NCsim hardware-level simulations (when the frame size is 176×144, search depth is 1 and bilinear interpolation is performed) the number of clock cycles to complete motion estimation for one frame is 2269396. It is relatively close to the estimated number of clock cycles according to the results of A|RT Designer. The difference lies in the actual implementation the VLIW at hardware level, where more cycles are required than A|RT Designer estimates.

C. Area Evaluation

For area evaluation, actual memory units are specified as 'black boxes', because they only have a VHDL-level description, which is dedicated only for behavioral verification and can not be utilized for efficient area analysis. Instead, area parameters of realistic memories with the same specifications are extracted from a database of CMOS12 technology. A fast single port SRAM memory module is utilized for the evaluation of the internal memory that stores a current macroblock. A fast dual port SRAM is utilized for the evaluation of the internal memory that stores reference macroblocks. A high-speed programmable ROM module is utilized for the evaluation of the instruction memory. Table II presents the number of resources that consumes the most amount of area in the implemented VLIW processor. Whereas, Figure 3 depicts the graphical view of the area contents.

[5]The height of a row, as well the width of a column, are 16 pels.

Resource	Area, μm^2	Percentage
Reference MBs' RAM	500000	44
Current MB's RAM	60000	5
Instruction ROM	92000	8
ALUs	189641	17
ACUs	9451	1
Multipliers	43670	4
Multiplexors	29830	3
Register files	178417	16
ROM	11958	1
Total area	1124731	

TABLE II
AREA CONTENTS.

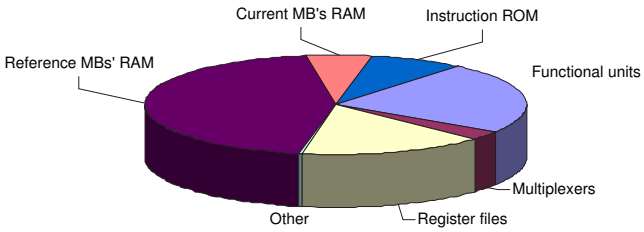

Fig. 3. Area contents.

V. CONCLUSIONS

Video encoding is usually an initial stage of many video processing applications found in consumer electronics such as digital video cameras, etc.. Within video encoding, motion estimation plays an important role being one of the most computationally intensive operation. In this paper, we presented an approach of mapping the motion estimation process to a VLIW processor template in

order to increase the performance of motion estimation. Furthermore, the approach is such that it still allows us to flexibly include support for other video encoding operations, e.g., DCT, IDCT, quantization, etc.. We have introduced new functional units and extended existing ones in order to include support for motion estimation. Furthermore, we have extended the instruction set with an instruction that initializes the sum of absolute differences operation, which is one of the most time-critical operation in motion estimation, on the functional units. The implemented VLIW processor is able to perform in real-time full search (search depth is 6, candidate motion vectors are in pel accuracy and frame rate is 30 fps) motion estimation of QCIF frame (176×144) with a clock speed at approximately 50MHz. The estimated area of the VLIW is 1.1mm^2. Furthermore, our solution is adaptable in the sense that other less compute-intensive search algorithms can be supported, such as three-step search, 2D logarithmic search, 3D recursive search.

REFERENCES

[1] S. Dutta A. Wolfe, J. Fritts and Edil S. T. Fernandes, *Datapath Design for a VLIW Video Signal Processor*, Tech. report, Princeton University, 1997.

[2] Adelante Technologies, *A|RT Designer Pro, User and Reference Manual*, v3.1 rev5 ed., 2002.

[3] Yu Hen Hu, *Programmable Digital Signal Processors*, Marcel Dekker, 2002.

[4] T-Chih Wang Jun-Fu Shen and Liang-Gee Chen, *A Novel Low-Power Full-Search Block-Matching Motion-Estimation Design for H.263+*, IEEE Transactions on Circuits and Systems for Video Technology, vol.11, no. 7 (2001), 890–897.

[5] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion estimation*, Kluwer Academic Publishers, 1999.

[6] K. Lengwehasatit, *A Novel Computationally Scalable Algoritm for Motion Estimation*, Visual Communications and Image Processing (1998), 66–79.

[7] D. Turaga M. Alkanhal and T. Chen, *Correlation Based Search Algorithms for Motion Estimation*, Tech. report, Carnegie Mellon University, USA, 1999.

[8] S. Reader and T. Meng, *Performance Evaluation of Motion Estimation Algorithms for Digital Signal Processors*, Tech. report, Stanford University, USA, 1999.

[9] N. Roma and L. Sousa, *A New Efficient VLSI Architecture for Full Search Block Matching Motion Estimation*, Tech. report, Instituto Supertor Tecnico, Portugal.

[10] P. Symes, *Video Compression Demystified*, McGraw-Hill, 2001.

[11] S. Wong, *Microcoded Reconfigurable Embedded Processors*, Delft University of Technology, 2002.

[12] Zhang Yong, *Novel Hierarchical Search Motion Estimation Algorithm for Mobile Video Transmission*, Tech. report, Nanyang Technological University, Singapore, 2000.

[13] Chi-Ying Tsui Zhong-Li He, Kai-Keung Chan and Ming L. Liou, *Low Power Motion Estimation Design Using Adaptive Pixel Truncation*, Tech. report, Hong Kong University of Science and Technology, 1997.