A Parallel Accelerator Architecture for Multimedia Video Compression

Bertil Schmidt and Manfred Schimmler

Institut für Datenverarbeitungsanlagen, TU Braunschweig,
Hans-Sommer-Str. 66, 38106 Braunschweig, Germany
{bschmidt, masch}@ida.ing.tu-bs.de

Abstract. This paper describes a parallel architecture for a variety of algorithms for video compression. It has been designed to meet the requirements of encoding and decoding according to the ITU-T standard H.263. The architecture is an implementation of the instruction systolic array (ISA) model which combines the simplicity of systolic arrays with the flexibility of a programmable parallel computer. Although the parallel accelerator unit is implemented on no more than 9 mm^2 of silicon it suffices to meet the compression rate necessary to send a compressed video stream through a standard ISDN terminal interface.

1 Introduction

Video compression for multimedia systems is a computationally intensive task. The desired compression factor determines the performance requirements of the underlying hardware. Several authors have suggested dedicated VLSI implementations [7] of video compression algorithms. Because of their regularity, high throughput rate, small silicon area, and low power consumption, systolic arrays have been proven as a good candidate structure for these dedicated solutions [2], [3]. Their disadvantage is the lack of flexibility with respect to the implementation of different algorithms and different problem sizes, i.e. each algorithm and each problem size requires an individual hardware solution. ISAs have been developed in order to combine the speed and simplicity of systolic arrays with flexible programmability [5]. Originally, the main application field of ISAs was supposed to be scientific computing. However, in the mid 90s the suitability of the ISA architecture for other applications was recognized, e.g. [6], [8]-[10]. In this paper we illustrate how an ISA architecture can solve all computationally intensive tasks of a multimedia video compression application efficiently. This ISA has been developed in order to compress a source data rate of 15 frames per second (fps) in CIF format according to the ITU-T standard H.263 [1] for a transmission with the given data rate of two ISDN channels.

This paper is organized as follows. The concept of the ISA is explained in Section 2. Section 3 gives an introduction to video compression with H.263. Section 4 presents the new video accelerator architecture. It is documented how the ISA is integrated on an accelerator for videophone applications. The parallel ISA algorithms

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 950-960, 1999.

for video compression are explained in Section 5 and their performance is evaluated in Section 6. The outlook to further research topics concludes the paper in Section 7.

2 Principle of the ISA

The ISA is a quadratic array of identical processors, each connected to its four direct neighbours by data wires. The array is synchronized by a global clock. The processors are controlled by instructions, row selectors and column selectors. The instructions are input in the upper left corner of the processor array, and from there they move step by step in horizontal and vertical direction through the array. This guarantees that within each diagonal of the array the same instruction is active during each clock cycle. In clock cycle k+1 processor (i+1,j) and (i,j+1) execute the instruction that has been executed by processor (i,j) in clock cycle k. The selectors also move systolically through the array: the row selectors horizontally from left to right, column selectors vertically from top to bottom. Selectors mask the execution of the instructions within the processors, i.e. an instruction is executed if and only if both selector bits, currently in that processor, are equal to one. Otherwise, a no-operation is executed.

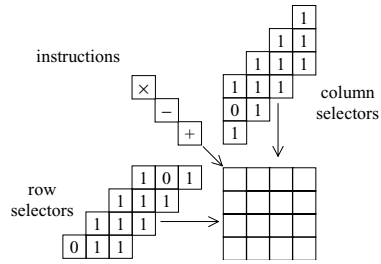

Fig. 1: Control flow in an ISA

Every processor has read and write access to its own memory. Besides that, it has a designated *communication register* (*C-register*) that can also be read by the four neighbour processors. Within each clock phase reading access is always performed before writing access. Thus, two adjacent processors can exchange data within a single clock cycle in which both processors overwrite the contents of their own communication register with the contents of the communication register of its neighbour. This convention avoids read/write conflicts and also creates the possibility to broadcast information across a whole row or column with one single instruction. This property can be exploited for an efficient calculation of row broadcasts, row ringshifts, and row sums which are the key-operations in many algorithms described in Section 5.

3 Multimedia Video Compression

The high amount of visual data associated with typical multimedia services establishes the need for efficient data compression schemes in order to facilitate transmission and storage applications. Several international standards have been introduced for video compression targeting different application fields. Communication applications (e.g. videophone, teleteaching) are covered by the ITU-T standard H.263. The H.263 codec codes video frames using a discrete cosine transform (DCT) on blocks of size 8 × 8 pixels. An initial frame is coded and transmitted as an independent frame. Subsequent frames, which are modelled as changing slowly due to small motions of objects in a scene, are coded efficiently in the inter mode using motion compensation (MC) in which the displacement of groups of pixels from their position in the previous frame are transmitted together with the DCT-coded difference between the predicted and original images. Fig. 2 shows a block diagram of the codec.

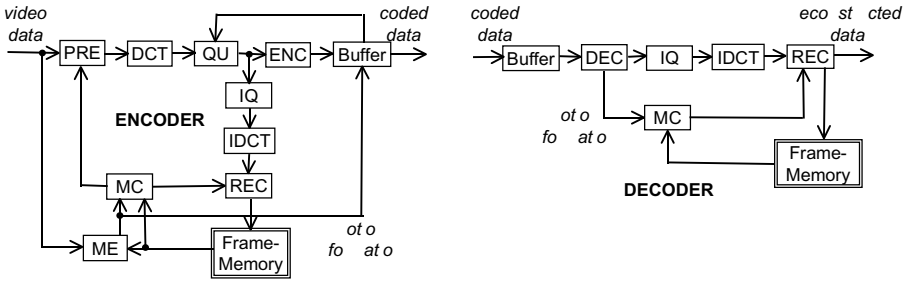

Fig. 2: Tasks and data flow of the H.263 encoder and decoder

The first step in the interframe coder is to calculate a motion vector for the current 16 × 16 pixels macroblock (**MB**) in ME (motion estimation). The motion vector is obtained by minimizing a cost function measuring the mismatch between a candidate MB in the previous frame and the current MB. Although several cost measures have been introduced [7], the most widely used one is the sum-of-absolute-differences (*SAD*) defined by $SAD = \Sigma_{k=0\ldots15}\Sigma_{l=0\ldots15} \mid c_{i,j}(k,l) - r_{i-u,j-v}(k,l) \mid$, where $c_{i,j}(k,l)$ represents the pixel (k,l) of a 16 × 16 MB from the current picture at the spatial location (i,j), and $r_{i-u,j-v}(k,l)$ represents the pixel (k,l) of a candidate MB from a reference picture at the spatial location (i,j) displaced by the vector (u,v). To find the MB producing the minimum mismatch error, we need to calculate SAD at several locations within a search window. The simplest, but the most compute-intensive search method, known as the full search or exhaustive search method, evaluates SAD at every possible pixel location in the search area. To lower the computational complexity, several algorithms that restrict the search area to a few have been proposed [3]. In baseline H.263, one motion vector per MB is allowed for motion compensation. Both, horizontal and vertical components of the motion vectors may be of half pixel accuracy, but their values may lie only in the ±15 range, limiting the search window used in ME.

The predicted MB represented by the calculated motion vector is loaded from the frame memory into MC (motion compensation). If the motion vector is of half pixel accuracy this operation requires an interpolation. The motion compensated prediction error is computed by the difference between the predicted and original MB in PRE (prediction). The resulting difference MB is transformed using a DCT of each 8×8 block, quantization by an adaptive quantizer (QU), entropy encoded using a variable-length coder (ENC), and buffered for a transmission over a fixed rate channel. The quantizer step size is calculated by evaluation of the buffer occupancy. After the quantization process, the original MB is reconstructed by the corresponding inverse operations (IQ, IDCT, REC) and stored in the frame memory. Of course, the reconstructed and original MB are not equal because of the performed lossy quantization. Since the reconstructed MB is available to encoder and decoder, it is used for prediction of the next frame.

The intra/inter mode selection is made at the MB level. If a MB does not change significantly with respect to the reference picture, an encoder can also choose not to encode it, and the decoder will simply repeat the MB located at the subject MB's spatial location. Only the processing instructions and capabilities of the decoder are standardized. The only necessary demand on the encoder is to produce a syntactically correct bitstream. The result is that the quality of H.263 video depends on the encoder implementation. In addition to the discussed baseline encoder algorithms, several negotiable options are offered by H.263 and H.263+. These optional modes allow developers to trade off between compression performance and computational complexity.

4 Accelerator Architecture for Video Compression

The analysis of the different video coding tasks and their processing efforts leads to a fixpoint processor architecture. The processor needs a small local memory for the storage and fast supply of local image data. The wordlength of the data items should be chosen to 8 bits because of the 8 bit input pixels and must also be able to process longer operands, e.g. DCT requires intermediate operands of length up to 24 bit. In order to achieve the real time requirements of the H.263 video codec several of these processors have to work in parallel. Thus, the particular processor has to be optimized with respect to its chip area and power consumption. Figure 3 depicts the processor architecture for the implementation of the H.263 video codec.

Due to the limited chip area the processor has to be very compact. In particular, it implies the choice of a bit-serial data organization. The size of the local memory is 32 internal registers plus 2 communication registers (C-registers). The word length of data items is 8 bits. Furthermore, each processor has two special constant registers 0 and −1. In addition to the registers, there are a zero flag and a negative flag that control the processing units depending on the state of the processor.

Two operand-multiplexers choose two registers of the own local memory or of the C-registers of the neighbours. The operands are propagated to the units required for execution of the current instruction. This units can be the multiplier and the

arithmetic, logical, and conditional unit (ALCU). The corresponding instruction set consists of 24 instructions. The result of the execution is given to the result bus and from there they are written into the local registers or the C-registers or both.

During the execution of one instruction the processor receives the next instruction together with a column selector bit from the upper neighbour and a row selector bit from the left neighbour. The selectors are interpreted. If both are 1 then the execution of the instruction is prepared. If one of the selectors is 0 then the execution of a no-operation is initialized. Instruction and column selector are propagated bit serially to the lower neighbour and the row selector to the right neighbour during the execution of the instruction.

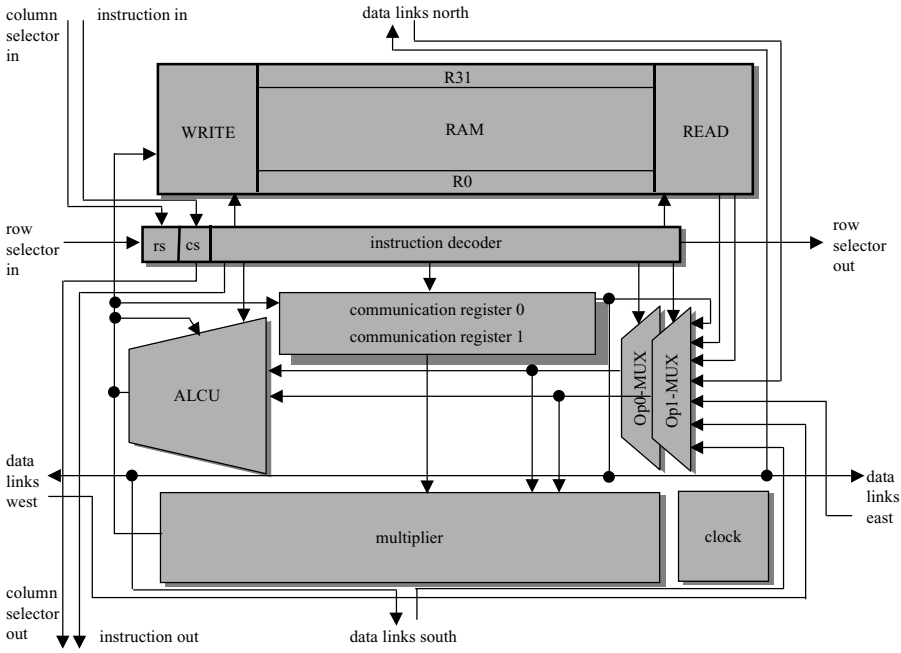

Fig. 3: Architecture of the array processor

The fine grained pipelined execution units are laid out for bitwise 300 MHz true single phase clock. Table 1 shows the areas of the different units of the processor. The full-custom design has been made with a 0.25µ digital CMOS process. An 12×8 ISA of the described processors has been structured in order to provide real-time processing for an H.263 codec at CIF resolution (352×288 pixels) and 15 fps. The resulting silicon area of the processor array is 3.7 mm^2. At a bitwise clock frequency of 300 MHz and a word format of 8 bits the theoretical peak performance of the array is 3.6 GIPS. The corresponding estimated power consumption is 120 mW. In order to exploit the computation capabilities of this unit, it is necessary to provide data and control information at an extremely high speed. Therefore, a cascaded memory

concept is implemented on-chip that forms a fast input and output environment for the parallel processing unit (see Figure 4).

For the fast exchange of data with the processor array each processor has two memory banks. Each memory bank contains 8 interface registers. One of these banks is always assigned to the corresponding processor, the other to the internal RAM by means of a fast data channel. The exchange of data between ISA and the internal RAM is done by bank switching. Both memory banks can be active at the same time, i.e. data transfer between ISA and internal RAM can be done concurrently to the execution of an ISA program. The internal RAM can also communicate bidirectionally with the external SDRAM.

The data transfer is controlled by an on-chip controller. The controller is started by the sequential standard RISC core (e.g. Hitachi SH4) and operates autonomously afterwards. It receives instructions from an instruction queue. The controller supplies the processor array with instructions and selectors that are stored in the ISA program memory. The instruction queue and the ISA program memory are located in the external SRAM, separated from the data. This part of the SDRAM is exclusively available to the controller by means of a fast channel. The internal RAM is organized as single ported static RAM. Its size of 8 KByte is mainly determined by the H.263 implementation (see Section 5). The layout of the complete video accelerator with a 0.25µ digital CMOS process requires an area of 8.1 mm^2 as depicted in Table 2.

Table 1: Area of the individual processor.

RAM	12.144 μm^2
Multiplier	6.398 μm^2
Read-write-logic	2.874 μm^2
ALCU	5.125 μm^2
Instruction decoder	4.011 μm^2
Addressing	1.190 μm^2
Routing	1.674 μm^2
Operand multiplexers	2.133 μm^2
Clock	2.613 μm^2
Sum	**38.135 μ^2**

Table 2: Area of the accelerator.

ISA	3.7 mm^2
Controller	1.1 mm^2
Internal RAM	3.3 mm^2
Sum	**8.1 mm^2**

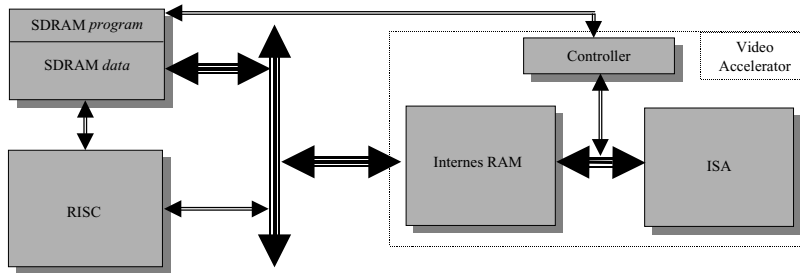

Fig. 4: Video accelerator architecture and data paths to the environment.

5 Parallelization of Video Compression Algorithms

The tasks of the hybrid coding scheme can be subdivided into low level, medium level, and high level tasks. Low level tasks typically operate on pixel data. They are highly regular and offer a large potential of data parallelism. Examples for low level tasks are full search motion estimation, DCT, and prediction. These tasks require a main percentage of the overall processing effort. Medium level tasks like quantization or entropy coding operate on the results of the low level tasks. Computation requirements are significantly lower than for low level tasks. The high level tasks comprise the bitstream handling and several control tasks, primary on encoder side, like quantizer control and decisions for coding strategy. High level tasks show an irregular control flow in conjunction with low computational rates. For control tasks a certain amount of flexibility with respect to modifications of the algorithms is advantageous. The proposed coprocessor architecture for multimedia comprise a flexible general-purpose RISC core with a less flexible accelerator adapted towards a specific type of processing. Thus, the RISC core performs the high level tasks of lower computational requirements, whereas the video accelerator executes the computation intensive but regular low level tasks and a part of the medium level tasks. The parallelization of the H.263 coding tasks on the proposed accelerator architecture is described in the following.

ME: In the 2D array structure of a 4×8 ISA the calculation of one SAD can be efficiently parallelized as follows: Assume, the current MB data $c(k,l)$ and the reference frame candidate MB data $r(k,l)$, $k,l = 0,\ldots,15$, are stored in the processor array such that processor (i,j), $i = 0,\ldots,3$, $j = 0,\ldots,7$, holds $c(4\cdot i,2\cdot j),\ldots,c(4\cdot i+3,2\cdot j+1)$ and $r(4\cdot i,2\cdot j),\ldots,r(4\cdot i+3,2\cdot j+1)$ in 16 internal registers. Now each node (i,j) computes the partial sum $\Sigma_{m=0,\ldots,3,n=0,\ldots,1}|c(4\cdot i+m,2\cdot j+n) - r(4\cdot i+m,2\cdot j+n)|$ within 30 instructions. The result is written into the two C-registers *C1* (most significant byte) and *C0* (least significant byte). The partial sums are added along each processor row within 2 instructions using two efficient row sum operations (see Section 2). The first operation adds up the values of *C0* and the second operation adds up the values of *C1* and the carry-bit of the previous addition. The total sum (SAD) for the candidate MB

is completed by the corresponding two column sum operations in the rightmost processor column and the result is written into *C0* and *C1* of the lower right processor. This SAD is compared in the lower right processor to the stored SAD resulting from the previous comparison. The smaller one as well as the corresponding motion vector is kept for future comparisons.

The preceding computation procedure is repeated until all possible candidate MBs are compared and the final motion vector is obtained. To speed up the computation the calculated SAD value is compared to a preselected threshold value. The search process terminates if the calculated SAD value is smaller than the threshold. Because the processor array of the video accelerator is of size 12×8 three different SAD values for the current MB can be calculated in parallel by a simple replication of the program for the 4×8 array.

Shifting of the candidate MB within the search area depends on the chosen search strategy. In order to avoid delay times in the systolic architecture it is necessary to know these locations in advance, and thus the next candidate MB data can be preloaded from the internal RAM into the ISA during the processing of the current MB. The ±15 pixels 2-step search with subsequent half pixel refinement is suitable and leads to a very good coding quality [3]. In the first step, it evaluates the motion vector candidates with distance two between two nearby search points within the ±15 pixels search area. Thus, 225 SAD values of previously known locations have to be calculated in the worst case. In the second step, the SAD values of the 8 neighbours of the calculated minimum point are evaluated. A subsequent half pixel search around the full pixel accurate displacement vector evaluates the motion vector with half-pixel accuracy. This involves the additional evaluation of 8 candidate blocks per motion vector. The locations of candidate points in the second iteration step and the half pixel search are data dependent. Thus, the data locations to be used at the next step are transferred to the controller when the current step is completed. At the beginning of the new step, the controller loads the corresponding reference data into the ISA. MBs of adjacent candidate vectors overlap quite significantly. In order to reduce memory accesses (bandwidth) only the non-overlapping data have to be loaded into the processor array. Since SAD values for 3 horizontally neighboured motion vectors are computed in the processor array in parallel only 2×20 pixels have to be loaded when the search area moves vertically up in the first step of the 2-step search. Search areas of adjacent MBs also overlap. Thus, only the non-overlapped data of size 48×16 pixels is loaded from the external SDRAM to the internal RAM, which can be performed concurrently to the motion vector computation of the current MB.

MC: The complete reference frame MB corresponding to the calculated motion vector has to be loaded into the ISA. In contrast to the other processing tasks ME is only performed on the luminance data of the MB. For MC and the preceding tasks another mapping of MB data onto the ISA is used. Each processor holds a 2×2 subarray. Thus, each of the six 8×8 blocks of the MB is mapped onto a 4×4 processor subarray of the 12×8 ISA. If the motion vector is of half pixel accuracy linear interpolation is processed in the processor array.

PRE/REC: The motion compensated MB and the original MB are simply subtracted (prediction) resp. added (reconstructed). Only 4 subtraction resp. additions have to be performed within each processor in parallel.

DCT/IDCT: The DCT of an 8×8 block $x_{i,j}$, $i,j = 0,\ldots,7$, can be expressed as

$$y_{k,l} = \sum_{i=0}^{7} c_{i,k} \left[\sum_{j=0}^{7} x_{i,j} \cdot c_{j,l} \right], \text{ where } c_{j,l} = \cos\left((2j+1)l\pi / 16\right) \quad (1)$$

The core operation in DCT computation is the multiply accumulation (MAC).

$$y_{2l} = \sum_{j=0}^{3} \left(x_j + x_{7-j}\right) \cdot c_{j,2l}; \; y_{2l+1} = \sum_{j=0}^{3} \left(x_j - x_{7-j}\right) \cdot c_{j,2l+1} \qquad \text{for } l = 0,\ldots,3 \quad (2)$$

Therefore, the multiplier of the introduced ISA processor has been designed to perform MAC efficiently, i.e. simultaneously with multiplication, a third operand can be added to the result without additional delay. As already indicated by the brackets in (1), the 2D transform can be decomposed into cascade of two subsequent 1D transforms applied horizontally and vertically on each row, resp. column. Additional efficiency can be gained, if we split the 1D transform of a sequence x_j, $j = 0,\ldots,7$, into even and odd numbered frequency samples:

This algorithm is applied for the 8×8 DCT within each 4×4 processor subarray. The input values for the 1D DCT are initially permuted within each processor row of length 4, such that each processor j, $0 \le j \le 3$, holds x_j and x_{7-j} and the addition $x_j + x_{7-j}$ and the subtraction $x_j - x_{7-j}$ can be performed in parallel. Now each processor j, $0 \le j \le 3$, computes y_{2j} and y_{2j+1} according to (2) in 4 stages. The computation in each stage requires two MAC operations per processor. Permutation of the values $x_j + x_{7-j}$ and $x_j - x_{7-j}$ between every two stages involves routing of data, which is performed efficiently within each processor row by ringshifting. The coefficient values are precomputed and loaded into the processor array when needed. This load operation takes only a small number of instruction cycles since the values can be broadcasted along the columns of the ISA. The 1D DCT along the processor columns is computed by the *reflected ISA program* for the rows. The IDCT is implemented in a similar way.

QU/IQ: The division operation for quantization is evaluated by multiplication with the reciprocal value in each processor in parallel. The quantizer step size is controlled by loading the corresponding coefficients into the processor array.

ENC/DEC: Due its irregularity the variable length en/decoding and entropy en/decoding is not suitable for an efficient parallelization on the ISA. Thus, the sequential RISC performs these operations. Because of their low computational requirements, their runtime on the RISC is dominated by the processing time of the next/previous MB on the video accelerator.

6 Performance Evaluation

For the runtime determination of the parallel programs described in Section 5 on the introduced accelerator architecture the number of required instructions is multiplied with the corresponding clock cycle time of 26.7 ns. Additionally, the data transfer

between ISA and internal RAM is considered with a throughput of 150 MByte/s. The results are shown Table 3. Note that the times for ME are worst cases, i.e. calculated SAD values are never smaller than the preselected threshold.

Table 3: Worst case runtime on the video accelerator for encoding and decoding per MB. It includes computing time on the ISA and data transfer time between ISA and internal RAM. Data transfer is divided into concurrent transfer, i.e. the transfer time is dominated by the computing time on the ISA, and not concurrent transfer, i.e. during the transfer time no computations are performed on the ISA.

Task		Runtime video accelerator	Data transfer ISA ↔ internal RAM	
			Concurrent to ISA	Not concurrent to ISA
Encoder	ME step 1	103 µs	Load 4080 Bytes	Load 640 Bytes
	step 2	6 µs	Load 36 Bytes	Load 288 Bytes
	half pixel	8.5 µs	Load 18 Bytes	Load 17 Bytes
	MC	5 µs	Load 128 Bytes	Load 451 Bytes
	PRE	0.5 µs		
	DCT and QU	10 µs		
	IQ and IDCT	10 µs	Store 384 Bytes	
	REC	3 µs		Store 384 Bytes
Decoder	IQ and IDCT	10 µs	Load 451 Bytes	Load 384 Bytes
	MC	2 µs		
	REC	3 µs		Store 384 Bytes
Sum per MB		161 µs	5097 Bytes	2584 Bytes
Sum per CIF		64 ms	2.02 Mbytes	1.0 MBytes

The amount of data transfer between external SDRAM and internal RAM per MB is

Encoder: current MB (384 Bytes), overlapping search area (768 Bytes), chrominance blocks of motion compensated MB incl. border pixels (162 Bytes), coded MB (384 Bytes), reconstructed MB (384 Bytes).

Decoder: current MB (384 Bytes), motion compensated MB incl. border pixels (451 Bytes), reconstructed MB (384 Bytes).

The resulting total amount of data transfer of 3301 Bytes per MB, resp. 1.3 MBytes per CIF frame, can be transferred concurrently to the accelerator activities with a reasonable throughput of the SDRAM bus, e.g. 100 MByte/s.

7 Conclusions

In this paper we have presented an ISA architecture for algorithms required for video compression. The accelerator unit has been implemented on an area of 8.1 mm^2 of silicon using a 0.25µ digital CMOS process. It is capable of on-line encoding and decoding of a video stream of 15 fps in CIF format with the H.263. The global architecture of the accelerator unit has been discussed as well as the detailed implementation of the single processing element of the 12 × 8 array. It has been

shown how the new architecture can be programmed for applications as motion estimation, motion compensation, DCT, and quantization. Apart from its performance figures, the most promising property of the accelerator unit is its flexibility. It would be interesting to study the performance for the new architecture for applications like [8], [9], [10].

References

1. Video Coding for low bitrate communication, ITU-T Draft Recommendation H.263, Geneva, May (1996)
2. Chang, Y.-T., Wang, C.-L.: New Systolic Array Implementation of the 2-D Discrete Cosine Transform and Its Inverse, IEEE Trans. Circ. Syst. Vid. Tech. 5 (2) (1995) 150-157
3. Cheng, S.-C., Hang, H.-M.: A Comparison of Block-Matching Algorithms Mapped to Systolic-Array Implementation, IEEE Trans. Circ. Syst. Video Tech. 7 (5) (1997) 741-757
4. Kunde, M., et al.: The Instruction Systolic Array and its Relation to Other Models of Parallel Computers, *Parallel Computing* 7 (1988) 25-39
5. Lang, H.-W.: The Instruction Systolic Array, a parallel architecture for VLSI, *Integration, the VLSI Journal* 4 (1986) 65-74
6. Lang, H.-W., Maaß, R., Schimmler, M.: The Instruction Systolic Array - Implementation of a Low-Cost Parallel Architecture as Add-On Board for Personal Computers, *Proc. HPCN 94*, LNCS 797, Springer Verlag (1994) 487-488.
7. Pirsch, P., Stolberg, H.-J.: VLSI Implementations of Image and Video Multimedia Processing Systems, *IEEE Trans. Circ. Syst. Video Tech.*, 8 (7) (1998) 878-891
8. Schimmler, M., Lang, H.-W.: The Instruction Systolic Array in Image Processing Applications, Proc. *Europto 96*, SPIE 2784 (1996) 136-144
9. Schmidt, B., Schimmler, M., Schröder, H.: Long Operand Arithmetic on Instruction Systolic Computer Architectures and Its Application to RSA cryptography, *Proc. Euro-Par'98,* LNCS 1470, Springer Verlag (1998) 916-922
10. Schmidt, B., Schimmler, M., Schröder, H.: The Instruction Systolic Array in Tomographic Image Reconstruction Applications, Proc. *PART'98*, Springer Verlag (1998) 343-354