Reiner W. Hartenstein, Jürgen Becker, et al.: A Reconfigurable Machine for Applications in Image and Video Compression; Conference on Compression Technologies and Standards for Image and Video Compression, Amsterdam, The Netherlands, March 1995

A Reconfigurable Machine for Applications in Image and Video Compression

Reiner W. Hartenstein, Jürgen Becker, Rainer Kress, Helmut Reinig, Karin Schmidt

University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de

ABSTRACT

This paper presents a reconfigurable machine for applications in image or video compression. The machine can be used stand alone or as a universal accelerator co-processor for desktop computers for image processing. It is well suited for image compression algorithms such as JPEG for still pictures or for encoding MPEG movies. It provides a much cheaper and more flexible hardware platform than special image compression ASICs and it can substantially accelerate desktop computing.

1. INTRODUCTION

Advances in digital technology leads to many applications in digital signal processing. Especially in image processing, large pictures or videos have to be stored efficiently. Modern image compression technology offers a possible solution. For still picture images JPEG [14] is used widely. Desktop publishing, graphic arts, colour facsimile, medical imaging are just a few areas where such compression algorithms are necessary. For compressing videos or general moving pictures the MPEG [4] algorithm or wavelets [6] are more appropriate. Doing the compression on desktop computers or workstations is normally to slow for high resolution applications. Specially the encoding of MPEG movies for e.g. CDI-ROMs etc. takes a lot of computing time. Several hardware manufacturers provide boards for such tasks, e.g. C-Cube [5] or Zoran [12]. These boards are very fast for a special type of algorithm. They allow to change some parameters to adapt the board on several image formats. But the algorithm cannot be changed. In this paper a reconfigurable machine for application in image and video compression is proposed. Due to the reconfigurability different algorithms can be implemented. This is much more flexible than usually used hardware compression boards, and faster than software solutions. The machine, called Xputer, can run stand alone or can be used as an universal accelerator on a host machine. The Xputer machine paradigm aims at the acceleration of algorithms, wherever the same operations are applied to a large amount of data. It describes the operation principles of a class of configurable hardware accelerators, which are based on field-programmable logic devices, so that the same hardware can be adapted to a lot of different algorithms. The resulting system is cheaper and more flexible than specialized hardware accelerators based on ASICs and much faster than software solutions on an unaccelerated workstation.

Section 2 introduces the Xputer machine paradigm and our prototype map-oriented machine 3. The usage of this accelerator is shown on the example of the JPEG compression algorithm in section 3. Performance results and a comparison with a modern workstation ends this section. How to encode video images with the MPEG algorithm is outlined in section 4. Finally some conclusions end the paper.

2. XPUTER

The term Xputer stands for a highly efficient non-von Neumann machine paradigm using a data sequencer instead of an instruction sequencer [8]. In contrast to data flow machines, where execution sequences are determined by arbitration, this novel data-procedural paradigm is fully deterministic. Because of only very loose coupling between sequencer and ALU, the Xputer paradigm supports high flexibility by using a reconfigurable ALU featuring efficient low level parallelism. The availability of several data sequencers running simultaneously is another source of parallelism. Compared to technologically equivalent von Neumann computers acceleration by up to three orders of magnitude have been obtained experimentally [8].

The key concept of an Xputer [2], [7] is to map the structure of performance critical algorithms into the hardware architecture. Performance critical algorithms typically apply the same set of operations to a large amount of data. Since ever the same operations are applied, an Xputer has a reconfigurable arithmetic-logic unit (rALU), which can implement complex operations on multiple input words and compute multiple results (figure 1). The large amount of input data typically requires an algorithm to be organized in (nested) loops, where different array elements are referenced as operands to the computations of the current iteration. The resulting sequence of data accesses shows a regularity, which allows to describe this sequence by a number of parameters. A simple example would be two nested loops, where the inner loop increments the column index to a two-dimen-

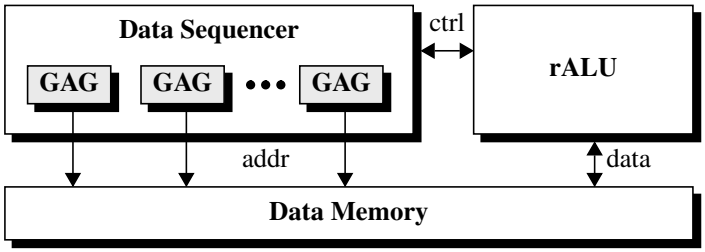

Figure 1. The Xputer paradigm

sional array, an the outer loop increments the row index, resulting in a video-scan like data access sequence. Such a sequence can be represented by seven parameters: the starting index, the stepwidth, and the maximum index, all three for both rows and columns, and a flag, which tells that columns have to be incremented before rows. An Xputer provides several hardwired Generic Address Generators (GAGs), which are capable of interpreting such parameter sets to compute generic address sequences to the data. Multiple Generic Address Generators simplify parallel access to different arrays of input data, which generally is the case for non-trivial algorithms. To be run on an Xputer, an algorithm has to be transformed into parameters sets for the Generic Address Generators and a configuration of the rALU, which implements the data manipulations within a loop body. Each time, the Generic Address Generators have accessed all input data of a loop iteration, the complex operations configured into the rALU are applied to the input data and the outputs can be written to the data memory by the Generic Address Generators. Since a compiler for an Xputer may rearrange the data in memory to optimize the data access sequences for the Generic Address Generators, a data map is required to describe the distribution of input and output data in the data memory.

The next section introduces an implementation of the Xputer paradigm, called MoM-3 (Map-oriented Machine) because it is the third prototype of a series of machines, which operate on a two-dimensional data memory.

2.1 MoM-3 overview

The Map-oriented Machine 3 (MoM-3) is the most recent implementation of the Xputer paradigm done at Kaiserslautern University. It supports up to seven Generic Address Generators, each with its own segment of data memory and a rALU subnet on the same board. The rALU subnets of all boards are connected to allow propagation of interim results to the next board (figure 2). That way complex operations, which require more resources than a single rALU subnet can provide, can be done on multiple boards. As long as the data required by a rALU subnet resides in the memory segment on the same board, data accesses can be done in parallel to data accesses on the other boards. Otherwise, data is transferred on the common MoMbus, which enforces a sequentialization of non-local data accesses. The (re-)configuration of Generic Address Generators and rALU subnets is done by a special MoM-3 controller circuit (M3C). It holds all configuration data for a complete application in its memory and is able to switch between configurations by downloading them to the Generic Address Generators or rALU

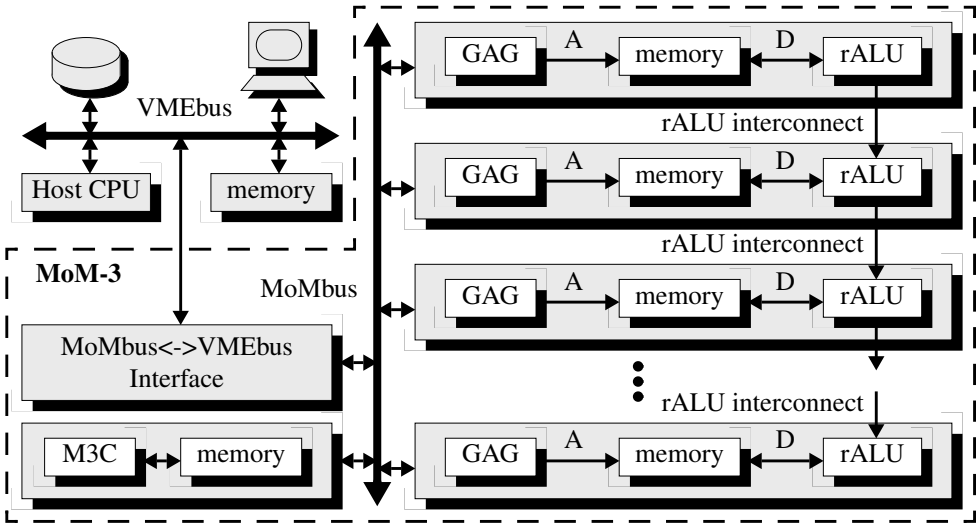

Figure 2. MoM-3 block diagram

subnets. The MoM-3 operates as a kind of configurable co-processor to a host computer. All I/O operations are done by the host as well as the memory management. The MoM-3 is a merely computational device to accelerate time-critical parts of algorithms. The host has direct access to all memory on the MoM-3, and on the other hand the MoM-3 can access data in the host's memory, though only sequentially due to the single bus in the host.

2.2 The Generic Address Generator

The Generic Address Generators in the MoM-3 operate in a two-stage pipeline. The first stage (handle address generator) which uses parameter sets to compute a handle position of a sliding window, providing access to a small region in the data memory. These sliding windows are called scan windows within the Xputer paradigm, because they are used to scan the input data. The second stage (memory address generator) computes an arbitrary sequence of offsets to the scan window handle position, to obtain the effective memory address for the data access (figure 3). The range of the offsets is bound by the size of the

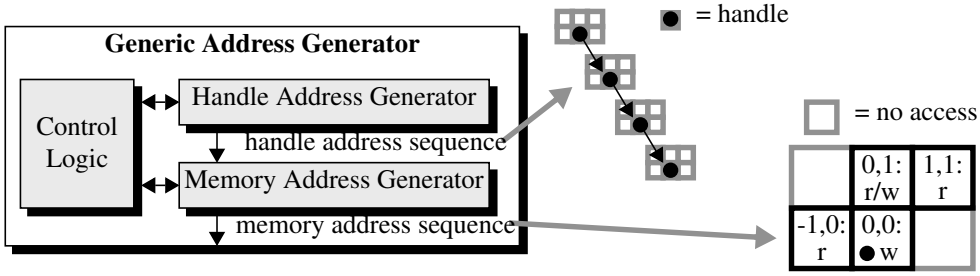

Figure 3. Generic Address Generator pipeline

scan window. The parameter sets, which can be interpreted by the handle address generator, result in a number of possible address patterns. These address patterns are called scan patterns because they describe the movement of the scan window. Figure 4 illustrates some examples of scan patterns, which can be done without requiring a reconfiguration by the MoM-3 Controller. Although most scan patterns are shown with a stepwidth of one, the stepwidth can be any size. The scan pattern most frequently used for the implementation of the JPEG algorithm is the basic video-scan, shown in figure 4b.

2.3 Reconfigurable arithmetic and logic unit

The reconfigurable arithmetic and logic unit (rALU) consists of a regular array of identical processing elements called datapath units (DPUs) [9]. Each DPU has two input and two output registers. The dataflow direction is only from west and/or north to east and/or south. The operation of the DPUs is data-driven. This means that the operation will be evaluated when the required operands are available. The communication between the neighbouring DPUs is synchronized by a handshake. To reduce the number of input and output pins, a serial link is used for data transfers between neighbouring DPUs on different chips. The DPUs belonging to the converters are able to perform their operations independent of the conversion. Using a serial link reduces the speed of the communication, but simulation results showed that by using pipelining, only the latency is increased whereas the throughput of the pipeline is decreased just slightly. Internally the full datapath width is used. For the

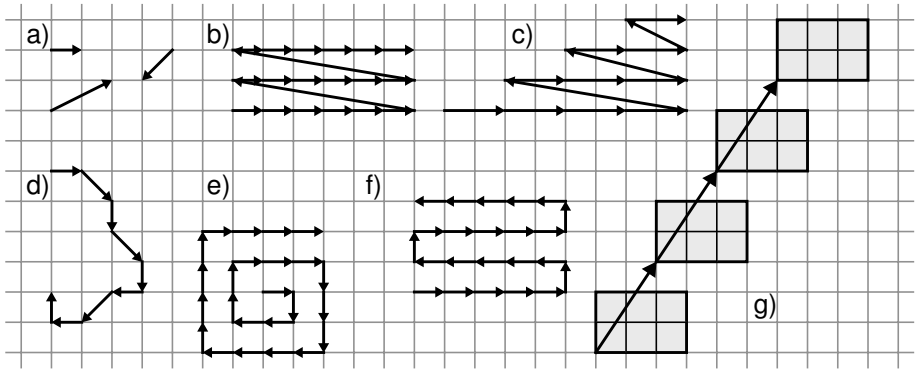

Figure 4. Scan pattern examples: a) single steps; b), c) video scans; d) data dependent scan; e) spiral scan; f) zig-zag scan; g) linear scan with scan window

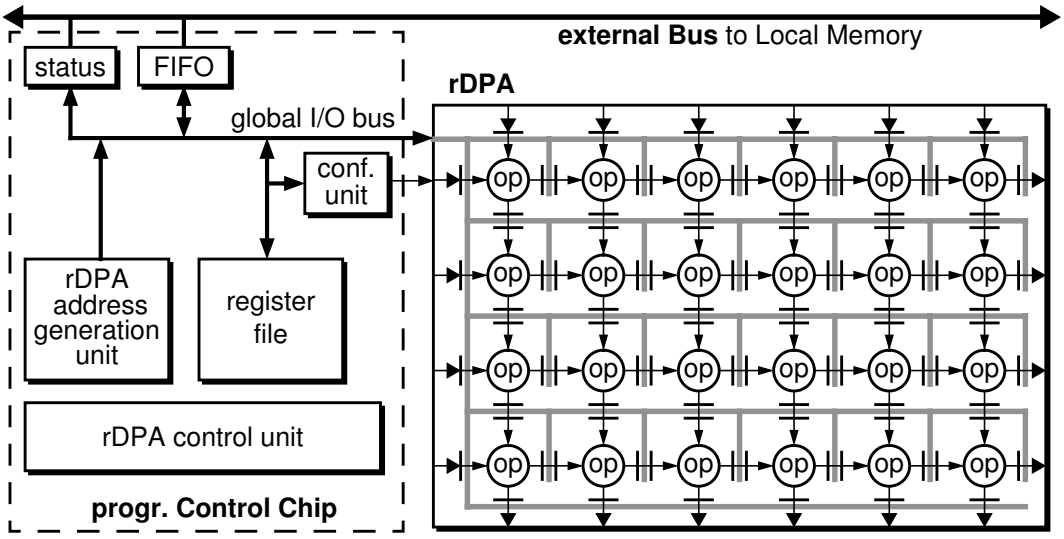

Figure 5. Reconfigurable datapath unit with programmable control chip

user and the software this serial link is completely transparent. A global I/O bus has been integrated into the rDPA, permitting the DPUs to write from the output registers directly outside the array and to read directly from outside. This means, that input data to expressions mapped into the rDPA does not need to be routed through the DPUs. The communication between an external controller, or host, and the DPUs is synchronized by a handshake like the internal communications. An extensible set of operators for each DPU is provided by a library. The set includes the operators of the programming language C. The operators of the DPUs are configurable. A DPU is implemented using a fixed ALU and a microprogrammed control. The configuration is data-driven, and therefore special timing does not have to be considered. With the proposed model for the DPA, the array can be expanded also across printed circuit board boundaries, e. g. with connectors and flexible cable. Therefore it is possible to connect the outputs of the east (south) array boundary with the west (north) one, to build a torus or to connect different rALU subnets.

Together with a programmable control chip the rDPA forms a data-driven reconfigurable ALU (rALU), as shown in figure 5. The control chip consists of a control unit, a register file, and an address generation unit for addressing the DPUs. The register file is useful for optimizing memory cycles. It makes it possible to use each DPU in the rDPA for operations by using the internal bus for routing. The address generation unit delivers the address for the DPU registers before each data is written into the rDPA over the bus. The rDPA control unit holds a program to control the different parts of the data-driven rALU. The instruction set consists of instructions for loading data into the rDPA array to a special DPU from the external units, for receiving data from a specific DPU, or branches on a special control signal from the GAG. The rDPA control unit supports context switches between three control programs which allows the use of three independent virtual rALU subnets. The control program is loaded during configuration time.

2.4 Programming

The MoM-3 can be programmed from a subset of C language called X-C [13]. The only features that are missing to obtain a full-featured C compiler are recursive functions and pointers, because both would require a dynamic memory management on the MoM-3 at run time. Output of the X-C compiler is assembly language for the rALU descriptions and the Generic Address Generators. These are translated by different assemblers to preserve the modularity of the hardware on the software side. The implementation of the JPEG algorithm is described below assembly language level most of the time, to make clear how the hardware actually processes the data.

3. JPEG

The JPEG standard includes two basic compression methods. A DCT-based method for lossy compression and a predictive method for lossless compression. The first one known as the Baseline method is considered here. Only the encoding is shown since the decoding is similar.

To increase the hardware usage the JPEG algorithm is implemented in four steps on our reconfigurable machine. This simplifies the explanation of the individual steps. The original picture coming from a host or a video interface is converted into a YUV image first. In this step additional functions such as gamma correction or filters easily can be implemented. Due to the reconfigurability of the Xputer it is simple to adjust the input interface to the different input formats such as 24-bit RGB, 16-bit YCb/YCr colour images or 12-bit greyscale images. The width of the communication buses in the rALU are always 32-bit. But operators can be adapted to compute with an accuracy of e.g. 8-bit or any other number. This improves speed for the compression algorithm since the image conversion is usually computed with 8 bit and the DCT with 11-bit. The full 32-bit I/O bandwidth is used at the rALU-memory interface. The dense packing of the data words is shown later in detail. An Y:U:V 4:2:2 sampling is included in the conversion. The second step is the two dimensional discrete cosine transform (2d-DCT) which is applied on an input of 8 by 8 blocks. The 2d-DCT can be computed by two 1d-DCTs:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{\cdot}\sum_{y=0}^{\cdot} f(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2x+1)v\pi}{16}$$

$$= \frac{1}{2}C(u)\sum_{x=0}^{7}\left\{\frac{1}{2}C(v)\sum_{y=0}^{7} f(x,y)\cos\frac{(2x+1)v\pi}{16}\right\}\cos\frac{(2x+1)u\pi}{16} \qquad (1)$$

$$C(u),\ C(v) = \begin{cases} \frac{1}{\sqrt{2}}; \text{ for } u, v = 0 \\ 1; \text{ otherwise} \end{cases}$$

The input data words are shifted from unsigned integer with range [0, 255] to signed integer with range [-128, 127]. The 2d-DCT computes 64 DCT coefficients, one DC and 63 AC coefficients. Because sample values vary slowly from point to point in an image, the compression can concentrate on the coefficients with lower frequency. The higher frequency coefficients have zero or near zero amplitude and need not to be encoded. Step three in our implementation is the quantisation. The goal of this step is to discard all the information which is not necessary to achieve the desired image quality. The 64 element quantisation table is configured as constants in the rALU. The last step is the coding of the DC coefficients and the run length encoding with the Huffmann coding. The necessary zig-zag sequence is generated automatically in the GAGs. The run length and the Huffmann coding is done in one step in the rALU. Figure 6 gives an overview of the different processing steps of the JPEG algorithm.

3.1 YUV conversion and block storage

The YUV conversion is done using three different areas of local memory, which can be accessed in parallel. The first memory area contains the original image in any colour model (RGB, CMYK, etc.), which can be represented by 32-bit words. The image is stored row by row in the two-dimensional memory. This is done by the host computer, either reading from disk, from a camera, or any other device which can serve as a source of such images. The layout of this input memory area, the so-called data map, can be seen in figure 7a. A row of 32-bit words stores one line of pixels as they would be displayed on a video screen. For example with the RGB colour model, the R, G and B values would consume one byte each, with the fourth byte of

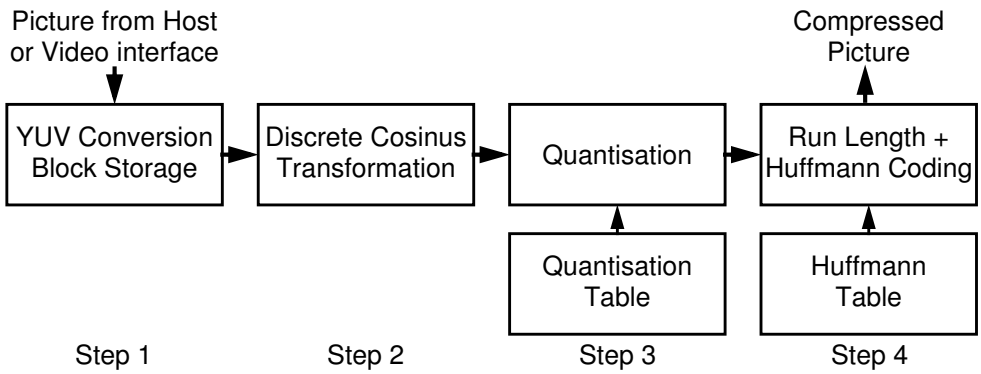

Figure 6. Overview of the different processing steps of the JPEG algorithm

the 32-bit word being left empty. The Y output is shown in figure 7b, where two adjacent Y values are packed together into one 32-bit word. This speeds up memory accesses, while retaining enough precision for intermediate results, which use only eleven bits of the sixteen available. The Y values are stored in blocks of eight by eight Y values, that is eight rows of four memory words. This is done because the following DCT step operates on two-dimensional eight-by-eight blocks of input values. That way, the first block of Y values corresponds to the first 64 pixels in the first line of the image. Due to the 4:2:2 sampling strategy, only half as many U and V results are produced. These are stored according to the data map in figure 7c. Each eight-by-eight block of U values is followed by a similar block of V values. Since every second U and V value is dismissed, the first U and V blocks contain the U and V values that correspond to the first 128 pixels of the first line of the image. Similar to the Y values, both U and V values are stored by packing two adjacent values into one 32-bit word.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}169 & -0{,}3316 & 0{,}500 \\ 0{,}500 & -0{,}4186 & -0{,}0813 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2}$$

The YUV conversion requires three Generic Address Generators (GAGs), see figure 8. The first reads the image in its original colour model from its local memory. The values are read in a simple video-scan pattern. In figure 8 this pattern is divided into 64-word linear segments only to visualize the relationship between such a 64-word segment and an eight-by-eight output block. The rALU performs the matrix multiplications to compute the YUV representation. The equations for an RGB to YUV conversion are shown in (2). Since the computations are quite simple, it would be possible to perform some filter operations on the input image before, resulting in a higher utilisation of the rALU resources. Furthermore, the rALU passes the computed Y, U and V values to the memory interfaces on the two following boards. On these boards, two other GAGs write the results back to their local memory. The first of these GAGs writes the Y values, using a block storage scheme. That is every 64 values are written to a block of eight lines of four 32-bit words each. Each 32-bit word contains two Y values of 16 bits, which is sufficient, since the following computations are done on 11-bit signed integers. The blocks are written line by line, then the next block follows with its first line. The third GAG performs a slightly modified scan pattern. Since the Y, U and V values are sampled at relative rates of 4:2:2, only half as many U values are produced compared to the number of Y values. The same is true for the V values. Therefore both the U and the V values are written to the local memory of the third GAG. The first value taken

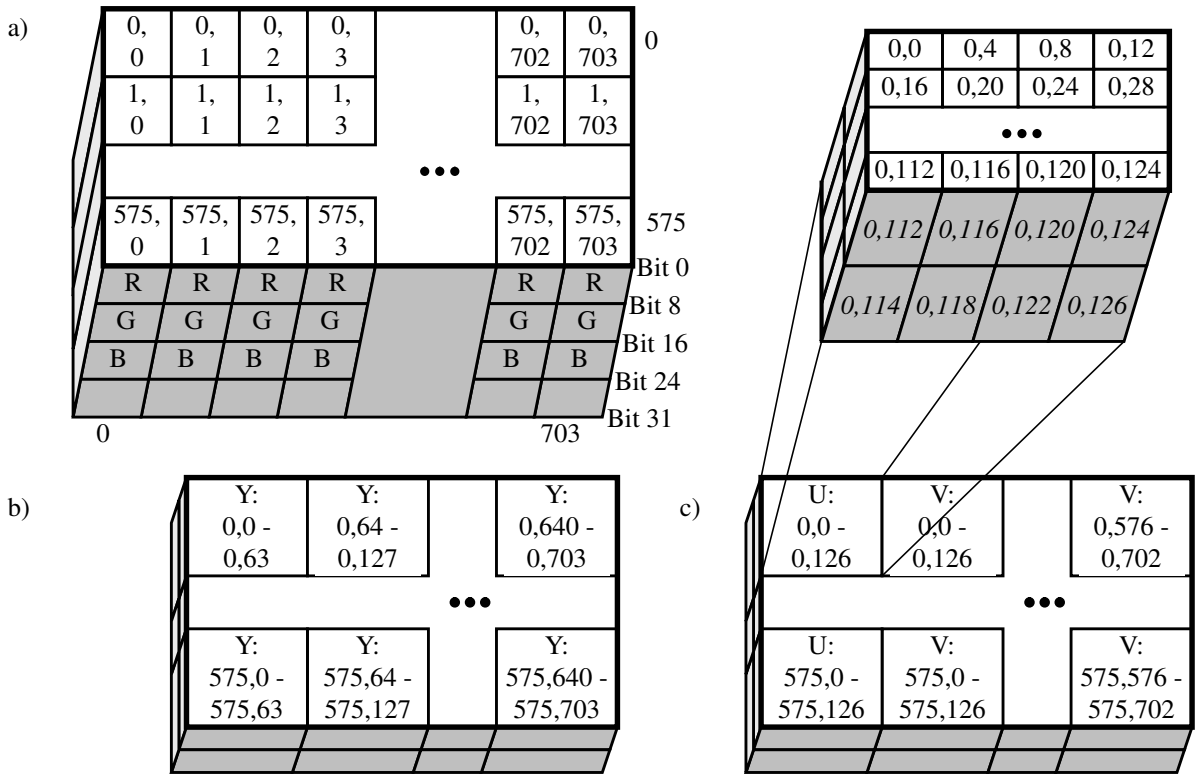

Figure 7. Data map for the YUV conversion: a) input data; b) Y output; c) U and V output

from the rALU is a U value, which is written according to the same block storage scheme as is applied to the Y values. The second value is a V value, which is stored with an offset of four 32-bit words to the right of the previously written U value, without continuing in the block storage scheme. That way two adjacent blocks of U and V values are written in an interleaved fashion while at the same time two blocks of Y values are written consecutively by the second GAG. After writing two blocks of U and V values, the third GAG skips the area of the block of V values, before starting with the next sequence of two U and V blocks.

At the end of the YUV conversion step, the whole image has been converted and distributed across two separately accessible memory areas. The Y, U and V values are stored in two-dimensional blocks of eight by eight values, which is exactly the data structure required for the two-dimensional DCT specified in the JPEG standard. The following computations can be accelerated further, if a total of seven boards with GAGs, memory and a rALU subnet are available. Three lines of the input image could be processed in parallel then. The first GAG would read three lines of the image in parallel. The rALU would distribute the first line to the second and the third board, like in the configuration above. The Y, U and V results corresponding to the second line of the input image would be stored on the fourth and fifth board, while the third line would go to the sixth and seventh board. Instead of computing the DCT only on the second and third board for the Y and the U/V values, respectively, a total of six boards could operate in parallel, resulting in a speed-up factor of three.

3.2 Discrete cosine transformation

The two-dimensional discrete cosine transform is done by applying two eight-point 1d-DCT to each eight-by-eight block of Y, U and V values. The first 1d-DCT is applied to the rows of the eight-by-eight block, while the second is applied to the columns of the result of the first DCT. In order to do both DCTs with the same configuration, the Generic Address Generators switch rows and columns when writing back the results. That way the second DCT may operate on rows like the first, because the contents of the rows are the columns then. When the second DCT has finished, rows are rows again and everything is fine. Since two values are packed into one 32-bit word, apart from the DCT operations, a pre- and a post-processing stage are required in the rALU. Both are shown in figure 9. The pre-processing stage unpacks the four values of a row in parallel and passes them to the DCT array. The unpacking is done row by row. The post-processing stage collects the outputs of every two rows and packs them as a single word to stored column by column. Each packed output word contains the values of two adjacent columns. The DCT operation in the rALU is pipelined, so that two rows of values are fed into the pipeline, before the first results can be read from the output ports.

The memory address sequence in the Generic Address Generators corresponds to the rALU pipeline in a way that two rows of four packed words are read from memory before the first packed results are written back. After eight rows have been read,

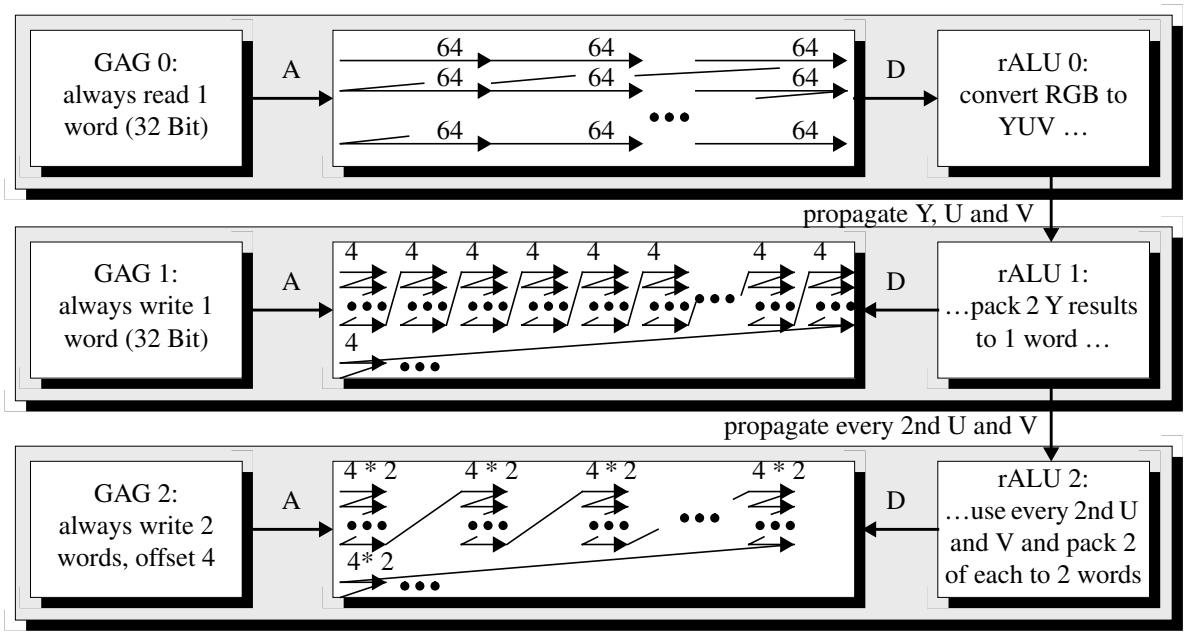

Figure 8. Scan patterns for the YUV conversion

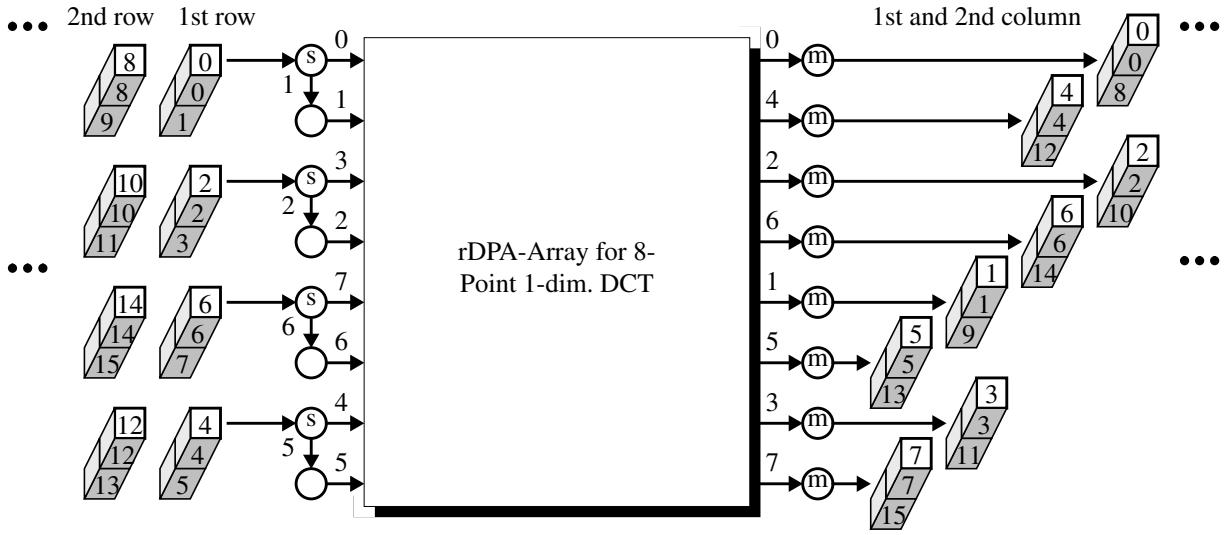

Figure 9. DCT pre- and post-processing stage in the rALU

the corresponding results are taken form the pipeline before the next eight-by-eight block is read. The results are written back in a quite complicated pattern, because both the row and column exchange and the bit-reversal addressing mode are combined. Figure 9 gives an idea of the read and write sequence. The inputs are read from top to bottom and from right to left from the positions corresponding to the values on top of the packed words. To obtain the position in the row, the values have to be divided by two and taken modulo 4. The outputs are written from top to bottom and from right to left to the first column of the row which is indicated by the value on top of the packed words.

The configuration of the rALU to compute the 1d-DCT is shown in more detail (including pre-and post-processing) in figure 10. For simplicity only the used communication paths of the 8 by 16 rDPA array are shown here. The implementation is based on Lee's algorithm [10]. The original flowgraph and the corresponding sparse matrix factors can be found in [11]. The split operators (s) and the merge operators (m) are the same as in figure 9. Two merge operators in sequence are used as kind of FIFO for storing the results which flush out of the pipeline. The multiplications with the constants are marked with a constant operator (k). The values can be found in [11]. The temporary variables (t#) which are produced in the DPUs are routed via the local register file of the rALU control to other DPU input registers.

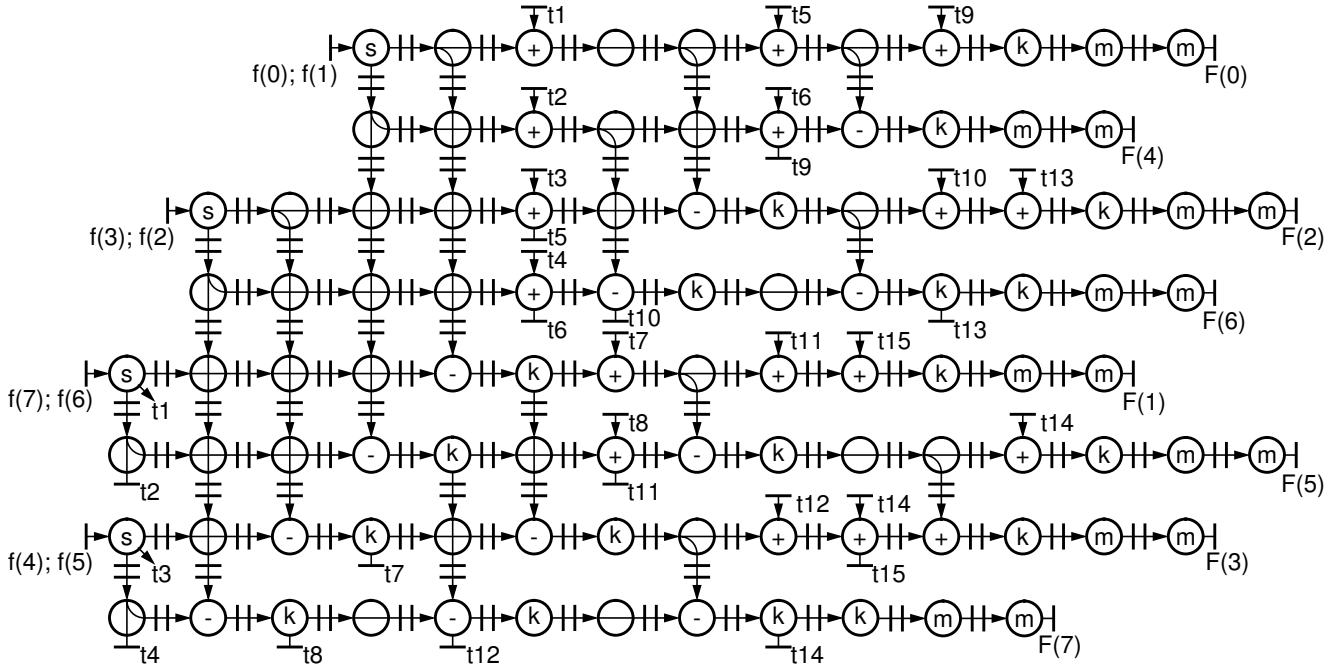

Figure 10. Configuration of the rDPA for an one dimensional DCT

3.3 Quantisation

Each of the 64 DCT coefficients is uniformly quantized in conjunction with a 64-bit quantisation table. The table must be specified by the user. There exists one table for the Y component and one for the U and V component of the image [3]. Quantisation including normalisation is defined as follows

$$F^{Q}(u, v) = \begin{cases} \left[\left(F(u, v) + \frac{Q(u, v)}{2}\right) \text{DIV } F(u, v)\right] \cdot F(u, v); \text{ for } F(u, v) \quad 0 \\ \left[\left(F(u, v) - \frac{Q(u, v)}{2}\right) \text{DIV } F(u, v)\right] \cdot F(u, v); \text{ for } F(u, v) < 0 \end{cases} \tag{3}$$

where F(u, v) is the original DCT coefficient, Q(u, v) is the quantisation constant and $F^{Q}(u, v)$ is the quantized DCT coefficient.

In each 32-bit word of one local memory segment two values of the Y component are packed. In a second memory segment one value of the U and one of the V component are packed. For each packed value the quantisation is done in two DPUs. The first DPU reads the packed two data words, unpacks them and writes one word to the second DPU. The first one computes the quantisation as noted in equation (3) and writes the quantized value to the second DPU. The second DPU starts with the quantisation and finishes with packing the two quantized values. These values will be written back to the local memory. For the 64 coefficients an array of 8 by 8 DPUs is required per subnet. The quantisation coefficients are directly configured as constants into the rDPA array. In figure 11 the quantisation is shown for two packed input words. The handle address generators of the GAGs perform a video scan over the beginnings of the blocks and the memory address generators produce the addresses of the packed data words.

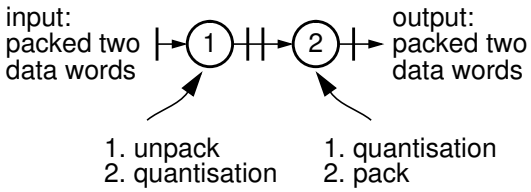

Figure 11. Quantization and normalisation of two packed input data words in two DPUs

3.4 Run length coding and Huffmann coding

The last step performs a run length coding on the quantized AC coefficients, followed by a Huffmann coding of the run length codes. The quantized DC coefficients are processed differently, in that the difference to the quantized DC coefficient of the previous eight-by-eight block is handed to the run length coding. The eight-by-eight block is read in a special zig-zag scan pattern for input to the run length coding (figure 12) [1]. This makes use of a property of the DCT to produce small values (often quantized to zero) towards the lower right corner of the eight-by-eight block. Long sequences of zeros can be expected

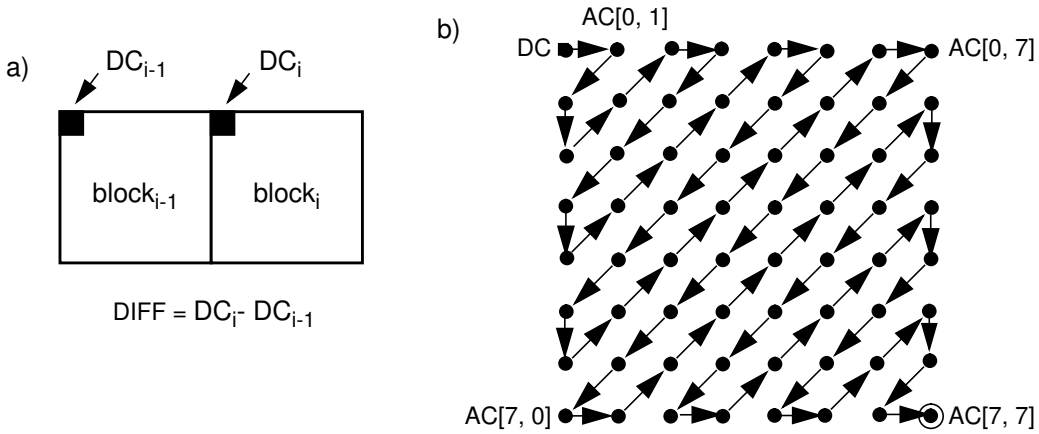

Figure 12. Preparation of quantized coefficients: a) differential DC encoding; b) zig-zag scan

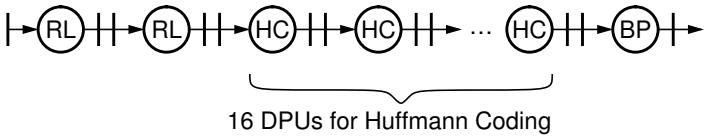

Figure 13. rDPA configuration for run length and Huffmann coding

in the input that way, resulting in high compression rates from the run length coding. This happens, because the run length coding produces an output value only for inputs different from zero, with a count of preceding zeros prepended. The rALU configuration requires two DPUs to perform the unpacking of memory words to two input words, DC coding and the run length coding. These DPUs are marked with RL in figure 13. Whenever an input to the run length coding is different from zero, two output words are produced. The first contains the run length code (RL code) with the number of leading zeros and the bit length of the following coefficient value different from zero. Special cases are the RL code for additional 16 leading zeros and the end-of-block (EOB) marker. The zero code is sent whenever the four bit zero counter overflows and is not followed by a coefficient value parameter. The EOB code is sent, when a zig-zag scanned block concludes with a sequence of zeros, and is not followed by a coefficient value parameter either. The next 16 DPUs perform the Huffmann coding on the output of the run length coding DPUs. Only the RL code words are translated with built-in Huffmann tables. The coefficient value parameters are passed unchanged. Each of the 16 DPUs contains the Huffmann table to all RL codes, which have the same zero count value. Output of the Huffmann coding DPUs is a stream of words of the following form: the first word contains two counters, one for the bit length of the Huffmann coded RL code, the second for the bit length of the optional coefficient value. The second word contains the Huffmann coded RL code and the third word contains the coefficient value, if necessary. The last DPU performs the bit packing to produce fully utilized 32-bit words. As long as the accumulated bit length of the Huffmann coded output does not exceed 32 bits, this output is shifted and OR-ed to the previous output. Whenever a 32-bit word is filled, it is written to memory and remaining bits of the output are put into the next 32-bit word. The last DPU maintains a bit counter for the bit length of the compressed result of this eight-by-eight block and a word counter for the number of 32-bit words already written to this block. Since the compressed output does not fill the eight by four memory words block, the remaining memory words are filled with zeros produced by the bit packing DPU (figure 14). Only the last memory word of the block is treated

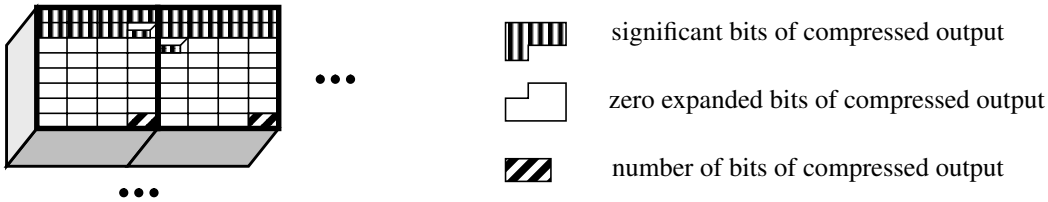

Figure 14. Data map of the JPEG compressed output

specially: it contains the bit counter, to indicate to the host, how many bits are significant in this block of eight by four memory words. Since the bit packing DPU fills the remaining words with zeros, the Generic Address Generators can simply read all 32 words of a block using the zig-zag scan pattern and write the 32 words in a simple video-scan pattern afterwards. The only thing that remains to the host computer is to scan all blocks of compressed output, read how many bits are significant and pack the segments of compressed output together before writing them to disk or another destination.

3.5 Performance

Table 1 shows some performance results on the JPEG algorithm. For the MoM-3 these results are given separately for the different parts of the algorithm. Two implementations are considered for the complete JPEG, one with three GAGs and one using all seven GAGs. The machine is implemented with a 1.0 m CMOS standard cell process and runs with 30 MHz. The performance figures are based on simulations. A complete memory access cycle can be done in 120 ns and a register file access in the rALU lasts about 60 ns. Compared to a modern workstation, a SUN SPARC10/51, the MoM-3 increases the performance by a factor of ten. Using the MoM-3 as an accelerator board for a desktop computer, the computer can run other tasks until the MoM-3 board signals the end of the compression with an interrupt.

processing step	number of necessary GAGs	number of active DPUs	performance (704 * 576 *24 bit RGB)
YUV conversion and block storage	3	59	0.8 ms
Discrete cosine transform	2	202	285.9 ms
Quantisation	2	128	48.7 ms
Run length and Huffmann coding	2	38	48.7 ms
Complete JPEG compression	3	202	384 ms
Complete JPEG compression using 7 GAGs	7	665	128 ms
SUN SPARC10/51, 192 MB RAM, 50 MHz	-	-	1300 ms

Table 1. Performance of the processing steps

4. MPEG

The MPEG coding algorithm also uses the above described methods of image compression from JPEG like discrete cosine transform (DCT), quantisation as well as run length and Huffmann encoding. So these steps of MPEG can be implemented on the Xputer as shown before. MPEG uses an additional technique in difference to JPEG, which is the method of motion compensation. Here the MPEG standard takes advantage of the temporal redundancy of previous or subsequent pictures within a video sequence.

4.1 Motion compensation

Motion compensation is a technique for improving compression by a factor of three compared to intra-picture coding. Motion compensation algorithms work at the macroblock level (16x16 pixel). The compressed file of a macroblock contains then the spatial difference (motion vectors) and the content differences (error terms) between the best matching block in the reference picture and the macroblock being coded. Motion compensation by macroblock-matching is widely used in today's high-compression video-coding and also in MPEG. Here for each current macroblock the best matching previous block is searched for, within a search area of 48x48 pixels. There are different criteria possible for comparing the content of two macroblocks. Here we use the L_1-distance of the pixel-values of the two macroblocks concerned.

$$L_1(\Delta i, \Delta j) = \sum_k \sum_l \left| X_t(k,l) - X_{t-1}(k+\Delta i, l+\Delta j) \right| \tag{4}$$

where (k,l) are the pixel coordinates within a macroblock and i and j are the components of the vector corresponding to the compared macroblocks. The motion vector $(i, j)^{opt}$ is determined by searching for a global minimum over all these distance measures within the search area.

$$(\Delta i, \Delta j)^{opt} = \min L_1(\Delta i, \Delta j) \; ; \quad \text{all } \Delta i, \Delta j \tag{5}$$

4.2 Implementation of motion compensation on the Xputer

In order to implement equation (4) and (5), the current macroblock, for which the best matching previous macroblock has to be searched, is always stored as a constant in the rDPA in a rALU subnet. For determining the best motion vector, a 16x16 scan window describes a video scan over the 48x48-pixel search area around the position of the current macroblock in the previous picture (figure 15). In every single step of the scan window the rALU subnet computes the L_1- distance (equation (4)). If the actual computed L_1-distance is smaller than the last stored minimum, a new minimum, a new error term and a new motion vector will be stored in the rALU. After the scan window has finished the video scan over the complete search area, the best matching previous macroblock is determined. Then the final error term and motion vector $(i, j)^{opt}$ (equation (5)) can be stored to be used in compression later on.

In order to increase the performance of this application, it is possible to divide the pictures into overlapping slices and to work with several GAGs and rALUs in parallel. So the Xputer can be well used as an off-line MPEG-encoder.

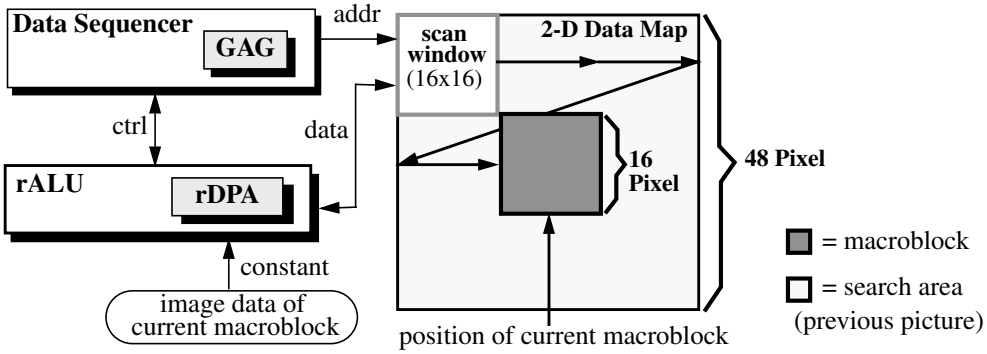

Figure 15. Xputer-configuraton for implementing Motion Compensation

5. CONCLUSIONS

In this paper a reconfigurable machine, called Xputer, has been presented for applications in image or video compression. Our prototype of the Xputer can be used stand alone or as an universal accelerator for image processing. The utilization of the machine has been shown for the JPEG image compression algorithm as an example for these kind of algorithms. Further the MPEG algorithm for encoding movies has been outlined. The Xputer provides a much cheaper and more flexible hardware platform than special image compression ASICs and it can efficiently accelerate desktop computers. A speed improvement over a modern workstation of a factor of ten can be obtained for JPEG image compression.

The prototype implementation Map-oriented Machine 3 has been completely specified with the hardware description language Verilog. The Generic Address Generators, the MoM controller and the programmable control chip for the rDPA have already been fabricated with standard cells during the European EUROCHIP project. An rDPA array will be submitted for fabrication soon. The programming environment has been specified and is currently being implemented on Sun SPARCstations.

6. REFERENCES

1. A. Ast, J. Becker, R. W. Hartenstein, R. Kress, H. Reinig, K. Schmidt: Data-procedural Languages for FPL-based Machines; Hartenstein, Servít (Eds.): Field-Programmable Logic, Springer-Verlag, Berlin, Heidelberg, pp. 183-195, 1994
2. A. Ast, R. W. Hartenstein, H. Reinig, K. Schmidt, M. Weber: A General purpose Xputer Architecture derived from DSP and Image Processing; in M. A. Bayoumi (Ed.): VLSI Design Methodologies for Digital Signal Processing Architectures; Kluwer Academic Publishers, Boston, London, Dortrecht, pp. 365-394, 1994
3. J. Buck: Hilfreicher Verlust; mc MicroComputer, pp. 94-101, Juni 1993
4. J. Buck: Komprimierte Bewegung; mc MicroComputer, pp. 114-123, April 1994
5. D. Bursky: Codec compresses images in real time; Electronic Design, vol. 41, no. 20, pp.123-124, Oct. 1993
6. P. Dessarte, B. Macq, D. T. Slock: Signal-Adapted Multiresolution Transfrom for Image Coding; IEEE Transactions on Information Technology, vol. 38, no. 2, pp. 897-904, March 1992
7. R. W. Hartenstein, A. G. Hirschbiel, M. Riedmüller, K. Schmidt, M. Weber: A Novel ASIC Design Approach Based on a New Machine Paradigm; IEEE Journal of Solid-State Circuits, Vol. 26, No. 7, July 1991
8. R. W. Hartenstein, A. G. Hirschbiel, K. Schmidt, M. Weber: A novel paradigm of parallel computation and its use to implement simple high-performance hardware; North Holland, Future Generation Computer Systems 7, pp. 181-198, 1991/92
9. R. W. Hartenstein, R. Kress, H. Reinig: A Reconfigurable Data-Driven ALU for Xputers; Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, Napa, CA, April 1994
10. B. G. Lee: A new algorithm for the discrete cosine transform; IEEE Transactins on Acoustic, Speech, and Signal Processing, vol. ASSP-32, pp. 1243-1245, Dec. 1984
11. K. R. Rao, P. Yip: Discrete Cosinus Transform: Algorithms, Advantages, Applications: Academic Press, Inc., Boston, 1990
12. A. Razavi, I. Shenberg, D. Seltz, D. Fronczak: A High Performance JPEG Image Compression Chip Set for Multimedia Applications; Proceedings of the SPIE - The International Society for Optical Engineering, vol.1903, p.165-74, 1993
13. K. Schmidt: A Restructuring Compiler for Xputers; Ph. D. Thesis, University of Kaiserslautern, 1994
14. G. K. Wallace: The JPEG Still Picture Compression Standard; IEEE Transactions on Consumer Electronics, Dec. 1991