Design and Implementation of the REMAP^3 Software Reconfigurable SIMD Parallel Computer

Lars Bengtsson[+] , Arne Linde[++], Tomas Nordström[++], Bertil Svensson[+], Mikael Taveniku[++], Anders Åhlander[+]

[+] @ Centre for Computer Science, Halmstad University, Sweden
[++] @ Division of Computer Science and Engineering,
Luleå University of Technology, Sweden

ABSTRACT

Highly parallel processor arrays will be useful components in advanced industrial real-time systems in the future. Especially in applications where the interaction with the environment utilizes visual, auditory, or tactile sensory units and advanced motor units massive parallelism is required. The possibility of making trainable systems by the use of artificial neural network (ANN) models is appealing in this context.

In REMAP^3, the Reconfigurable, Embedded, Massively Parallel Processor Project, the focus is on the design of processor array modules and the mapping of ANN computations on these modules. The paper describes the structure of the first prototype which utilizes dynamically programmable logic cell arrays to allow architectural experimentation with the modules. Examples of processing element architectures are given. Low-level programming is described.

Presented at DSA -92, the Fourth Swedish Workshop on Computer System Architecture, Linköping, Sweden 13-15 January 1992

1 THE REMAP[3] PROJECT

REMAP[3], the Reconfigurable, Embedded, Massively Parallel Processor Project, is a joint effort between the Centre for Computer Science at Halmstad University and the Division of Computer Science and Engineering of Luleå University of Technology. It is based on the conviction that computing systems for tomorrow's advanced interactive applications, incorporating visual, auditory, or tactile sensory units and advanced motor units, will require massively parallel processing capabilities and new, parallel I/O interfaces. Special attention in the project is given to the possibility of executing massively parallel learning algorithms, making the total system trainable and giving it an ability to show adaptive behaviour.

With this point of view, massively parallel processing is now entering the realm of embedded systems, which, in turn, implies that considerations like application tuned architectures, special purpose I/O processing, and real-time demands are placed in focus.

An architecture based on the paradigms of massively parallel SIMD computing, application specific, I/O close processing, and trained behaviour may look like the one shown in Figure 1. It would execute programs in a cyclic fashion, with data constantly flowing between the modules.

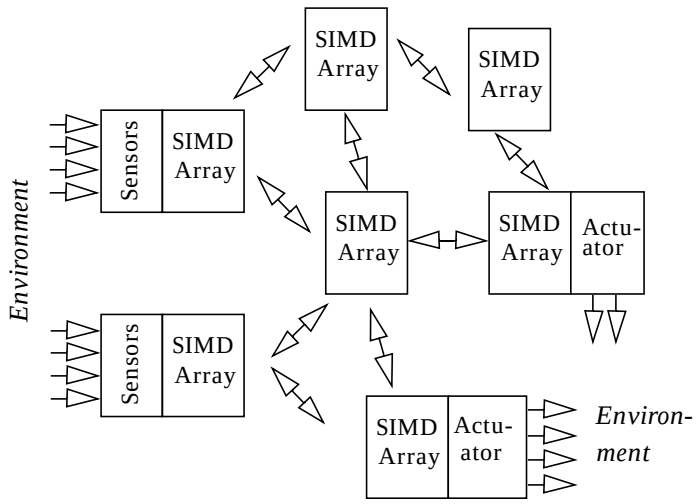

Figure 1 An embedded, massively parallel, modular system.

The REMAP[3] project aims at obtaining a massively parallel computer architecture put together by modules in a way which allows the architecture to be adjusted to a specific application. This suggests that a certain architecture may be"compiled", thus a modification of each module and adjustments of the connections between the modules are enforced.

Within the project, a series of studies has been performed [Svensson, 1989], [Svensson and Nordström, 1990], [Nordström, 1991a, 1991b], [Nordström and Svensson, 1992] concerning the execution of Artificial

Neural Network (ANN) algorithms on highly parallel SIMD computers, with special emphasis on architectures based on bit-serial processing elements (PEs).

The results show that SIMD is the best suited parallel processing paradigmot today for artificial neural networks and that the demands on the inter-PE communication can be met with surprisingly simple means. Arrays of bit-serial PEs can be very efficiently used for ANN computations. Since multiplication is the single most important operation in these computations, there is much to gain in the bit-serial architecture if support for fast multiplication is added. This will be demonstrated in the PE design for REMAP^3 shown below.

REMAP^3 is an experimental project. A sequence of gradually evolved prototypes is being built, starting with a small, software configurable PE array module, implemented as a Master's thesis project [Linde and Taveniku, 1991]. With only slight modifications in PE array architecture, but with a new high-performance control unit, the second prototype is now being built, almost full-scale in PE number, but far from miniaturized enough for embedded systems.

The early prototypes rely on dynamically programmable logic cell arrays (FPGAs). Therefore, different variations of the prototypes can be realized by reprogramming. The FPGAs are designed for high speed. Thus, the speed and the logical size of the prototype systems suffice for new, demanding applications, but the physical size does not allow embedded multi-module systems to be built from the prototypes.

Based on the experiences from the FPGA based prototype modules, a proposal for a VLSI implemented module will be made, as well as for a system architecture for a heterogeneous system of modules.

Switching to the paradigm of trainable systems and modular, massively parallel computing, tightly coupled to sensors and actuators, will most certainly call for new application development methods. In connection with REMAP^3, ideas on a programming environment for the development of trainable integrated systems are being formulated [Wiberg, 1991].

2 THE REMAP^3 SIMD COMPUTER PROTOTYPE

The logical structure of the parallel computer is shown in Figure 2. The machine is a *SIMD (Single Instruction Multiple Data)* organized architecture meaning that a large number of *Processing Elements (PE's)* are working in parallel with different data, but all doing the same thing on this data. It is the task of the *Control Unit* to broadcast the microinstruction (control and address) plus the system clock to all PEs in the system. The micro instructions are very low-level such as ADD, SUBTRACT, MOVE, BROADCAST etc. in each clock cycle. Thus, the Control Unit has complete control of all actions in the system. This means that the machine is synchronizedon very low level; therefore no synchronization problems occur when different actions need to be done in a specific order.

The PE's forms an array of processors which today is linearly connected. Each PE may communicate with two of its nearest neighbours (NORTH,SOUTH). Also, *broadcast* is possible where one PE may send information to all PE's in the system. This feature is very useful in neural network execution. Each PE has its own memory (256 k * 1 bit) and the mode of operation is *bit-serial* meaning that each operation (microinstruction) operates on one or two bits of data only. I/O is handled with a *corner-turner* section which transforms the byte parallel data from the outside world to the bit serial world of the array.

The Reconfigurability of the PE-array has been accomplished by using *FPGA's (Field Programmable Gate Arrays*) that are downloaded with configuration data whenever the PEs need to be reconfigured. Different configurations may thus be used for different applications and algorithms. One configuration may suit a specific algorithm in image processing, another may suit some special neural network execution and a third may be used for a signal processing application. Detailed research on the perfect combination of architecture/algorithm is thus possible with this prototype.

At the top, a *Master Processor* controls it all. All I/O, FPGA configuration downloading and instruction initiation are handled by this part. In the implementation, a conventional microprocessor (the 68000) has been used for this purpose. A link to a Host computer (Workstation or PC) enables the use of convenient environment for software development for the parallel array (microprograms in low- and high-level descriptions).

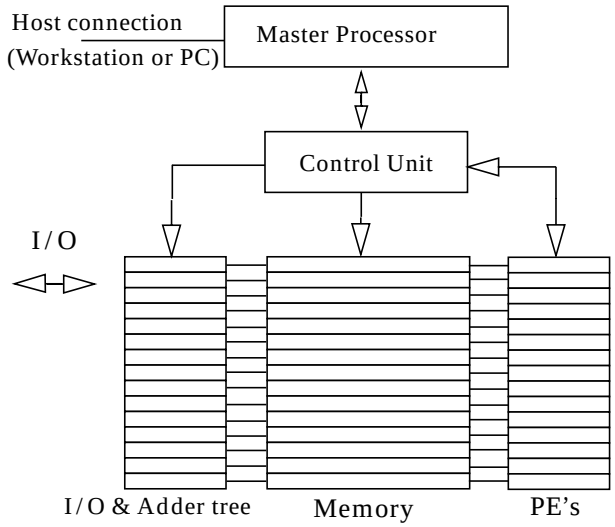

Figure 2 Logical structure of the REMAP^3 SIMD computer

3 THE CONTROL UNIT (CU)

The principal task of the Control Unit [Bengtsson,1991] is to generate microinstructions and the global system clock to the processor array. Each microinstruction consists of an *address part* (max. 32 bits) and a

control part (max. 32 bits). The address is generated by the *Address Processor* part and the control field by the *Controller* part. This Controller is a microprogrammed one-stage pipeline running at max. 10 MHz, generating a new microinstruction *every 100 nanoseconds*. The microword is 96 bits wide, 32 bits controlling the array and 64 bits controlling the CU itself.

The CU interfaces to the Master Processor through the *MP Interface*. The VME bus standard with 16-bit transfers is used for the connections of all cards in the system.

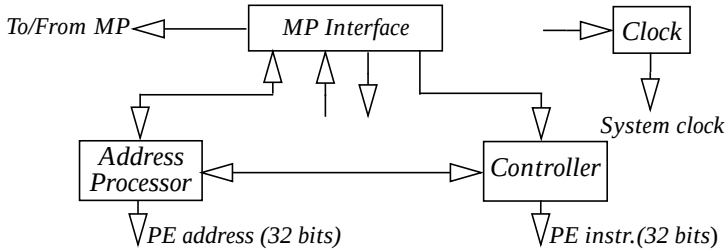

Figure 3 The Control Unit

The Controller part sequences the microprogram execution by means of a AM29C331 sequencer chip. This can be thought of as an advanced program counter, including a stack for subroutine calls, a counter register for looping, a breakpoint register and more. The principal output of this chip is a 16-bit address delivered to the microprogram memory.

The microprogram memory (the *Writable Control Store - WCS*) is an 8k deep and 96 bits wide static CMOS storage. Microprograms are downloaded from the Master Processor to this memory under complete software control. The 96 bits of output is fed directly to the *pipeline-register*, which holds the current microinstruction while the next one is being calculated in the pipeline.

The Address Processor is composed of three major parts – the *Register File*, the *ALU* and the *Address Register*. The first one contains 64 registers with 32 bits each, constructed from two AM29C334 chips. This part is actually shared with the MP Interface in such a way that the MP *writes* in the file, and the Address Processor *reads* from it. This also facilitates parameter passing between the MP and the parallel computer.

The ALU is the AM29C332 chip, capable of full 32 bit arithmetic and logical operations.

The Address Register serves the purpose of a pipeline-register in the data-path. It holds the current address while the next one is being calculated by the Address Processor.

An important feature of the address processor is its ability to do a *read-modify-write* operation in one clock cycle (100 ns). This is important for the sake of performance, especially since the PE's work bit-serially.

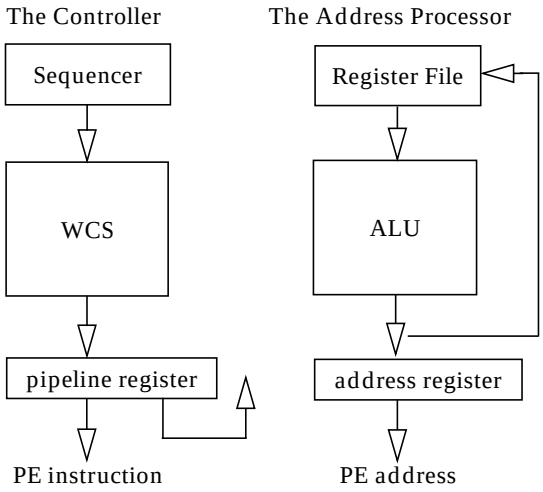

Figure 4 The Controller and the Address Processor

4 THE PROCESSOR ARRAY

In the project, two steps are taken in the development of the final complete machine. In the first step, the *alpha* version is implemented consisting of the Control Unit Card, the Master Processor Card and the LUPUS Processor Card. LUPUS has been designed and implemented as a Master's thesis project at Luleå University of Technology [Linde and Taveniku, 1991]. The structure of it is shown in Figure 5.

The FPGA's on this card hold 12 PEs of the type depicted in Figure 6. Each PE has a static storage of 256k * 1 bit. The PEs are linearly connected (nearest neighbour communication) as well as connected to a common broadcast bus. This facilitates "one-to-all" transmission, useful e.g. in neural network execution. I/O is channelled through a corner-turner (CT) section (also implemented with FPGA's), transforming the byte parallel data from the outside world to the bit-serial world of the array.

The LUPUS processor card is capable of running at 3 MHz.

The second step to the final processor array (the *beta* version) will utilize the latest FPGA family from XILINX – the X4000. These circuits will permit more PE´s to be implemented in each chip, giving at least 32 PE´s on each card. Eight such cards will give 256 PE´s in total.

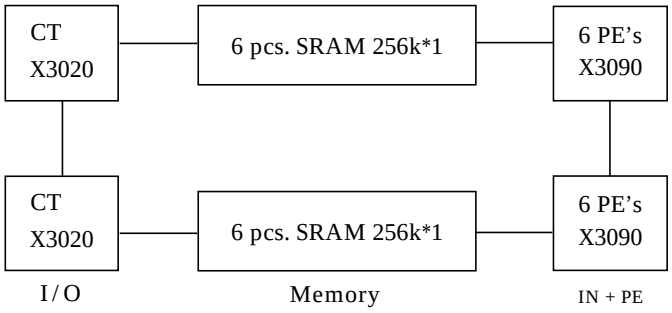

Figure 5 The LUPUS processor card.

4.1 Processing Element

As shown in Figure 6 the PE is divided into a number of sections. This is reflected in the µ-code as different fields which are defined as InMux,CarryMux,RMux,TagCtl and Multiplier. The C (carry) flipflop is mostly a carry register but it can also be used as a temporary storage, for example in compare and search instructions. The T (Tag) register is used as an activation control register but it can also be used as a temporary storage. The R/W or T control determine whether data is to be written to memory or not, this is controlled by the tag control section of the micro word

.

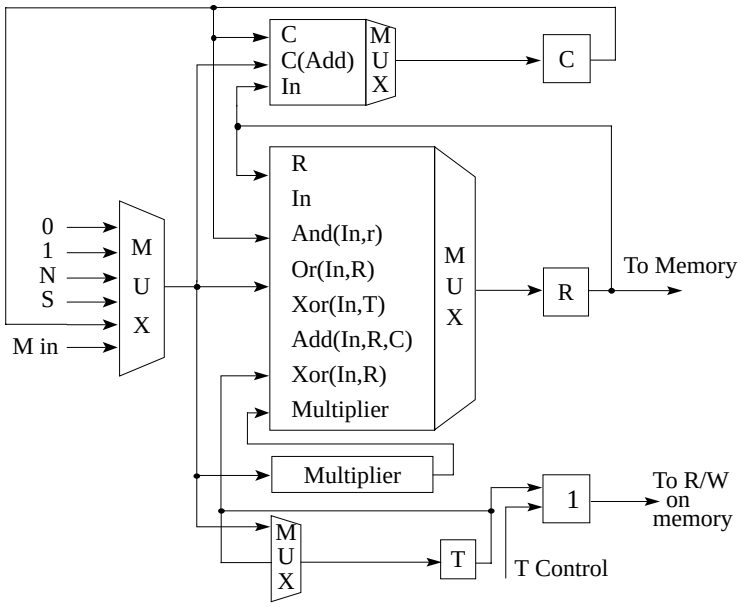

Figure 6 The LUPUS PE

4.2 The Corner Turner

The Corner Turner (CT) performs the transformation between a conventional byte oriented I/O format and the internal bit-slice format. This is obtained by a two dimensional shift operation as shown in Figure 7. In the vertical shift mode MP shifts in and out data at the top and bottom, respectively. During this time the PEs can perform computations in the PE memories. Transfer between CT and the PE memories is made by a horizontal shift, i.e. one bit-slice is transferred each cycle. Because of their very simple structure, the corner turners are capable of running much faster than the PE's (speed up to 40 MHz) thus enabling fast input and output to the PEs.

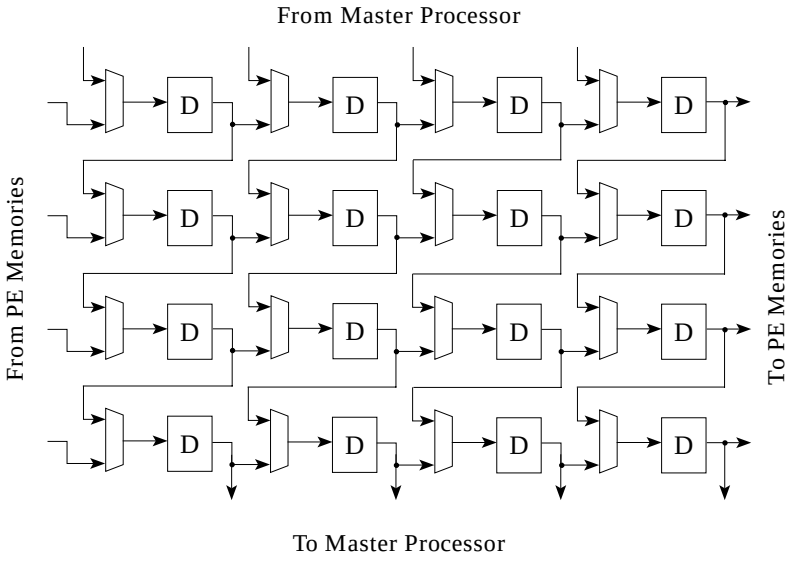

Figure 7 Corner Turner principle

4.3 Xilinx Programmable Gate Arrays

The structure of the Xilinx circuits is shown in Figure 8. The chip consists of a number of Combinatorial logic blocks (CLBs), some input-output blocks (IOBs) and an interconnection network (ICN). These circuits are user programmable, thus enabling the CLBs, IOBs and ICN to be programmed by the user. This is accomplished at power up by a reprogramming sequence that loads the on-chip RAM with the configuration. The RAM can be loaded from an external memory or from a microprocessor, the latter is used for LUPUS. It takes about 200ms to reprogram the circuits.

The IOBs connect the I/O-pads to the ICN; these blocks can be configured as input, output or bidirectional blocks. The CLBs are configurable logic blocks consisting of one 32 bit lockup table for logic functions and two flipflops for state storage, these blocks are only connected to the ICN.

The ICN connects the different blocks in the chip, it consists of four kinds of connections: short-range connections between neighbouring blocks, medium-range connections connecting blocks on slightly larger

distances, long-lines connecting whole rows and columns together, and a global nets for clock and reset signals broadcast throughout the whole chip.

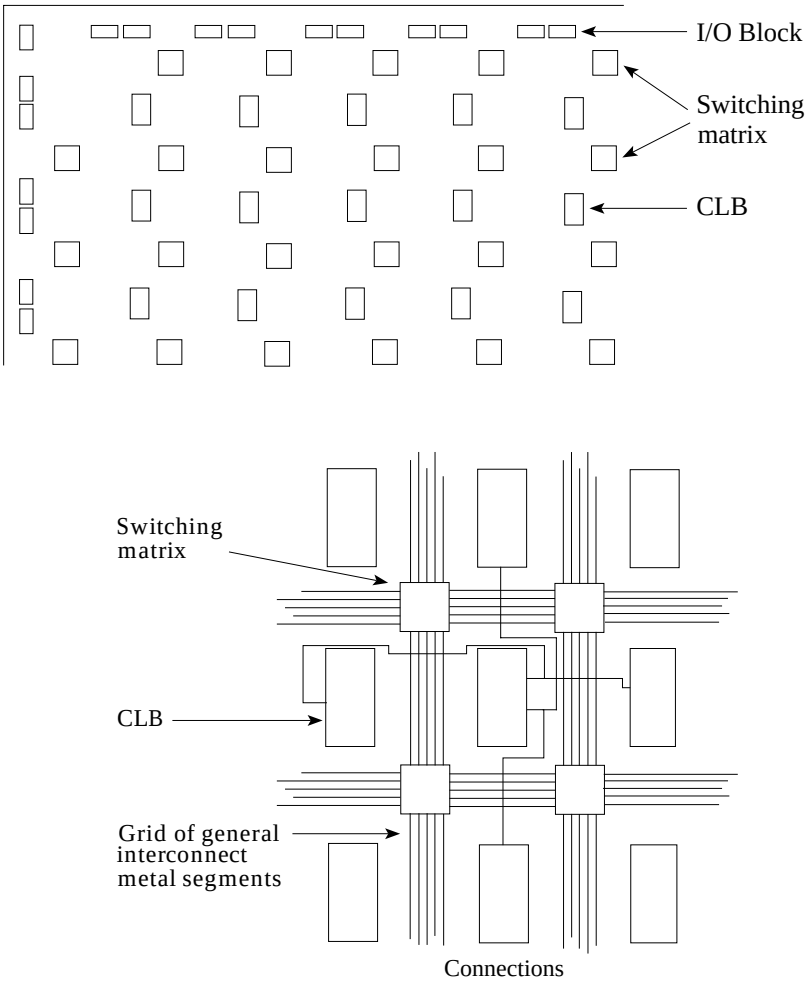

Figure 8 Xilinx XC3000 overview

Figure 9 shows the structure of the IOBs and CLBs.

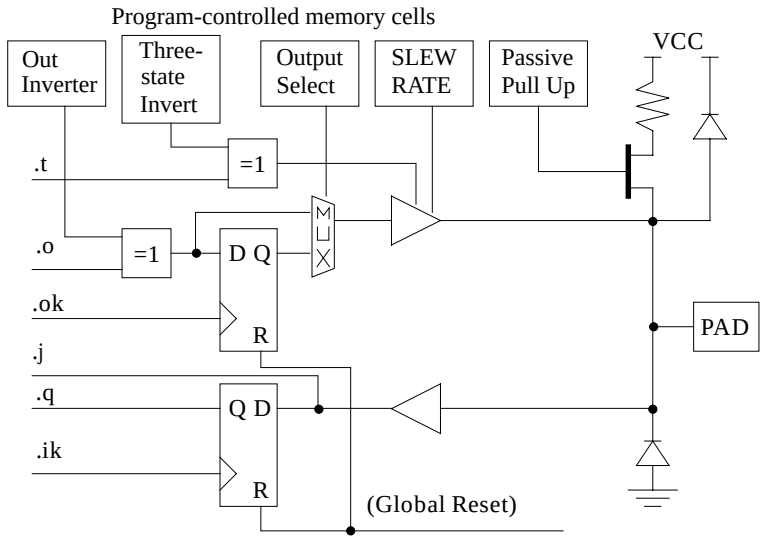

Figure 9 XC3000 I/O Block and Combinatorial Logic Block

5 PROGRAMMING

Program development is (or will be) supported at various levels. Microprograms, i.e. the programs in the WCS addressed by the sequencer, is the lowest level of program on a REMAP^3 computer. As a first step we have implemented a low-level language that we call MASS, described in section 5.1. A future higher-level language for microprogramming is planned, and is outlined in section 5.2 below.

Above the microprogramming level, i.e. in the Master Processor, there will be a data-parallel language like C* [TMC, 1990] or Parallaxis [Bräunl, 1989] and we are also considering using application specific languages like Pygmalion [Treleaven, 1991] for artificial neural networks programming.

5.1 Low-level micro-programming. The MASS microcode assembler.

The MASS Microcode ASSembler is a general purpose tool for writing microprograms for microprogrammed hardware. A 'FIELD' statement in the source code informs the assembler about an interval of bits in the microword that should be assigned a specific name, and what mnemonic (or numeric) codes that this field can have. A leading asterisk ('*') on one mnemonic defines the default value for this field, if other than the first one. Numeric values are assigned to each mnemonic by the assembler starting with zero (if no initializer is explicitly given) at the left-most position and then by increments (by one) to the right in the list.

The 'INCLUDE' statement is used to specify a file which should be included in the source text flow. Commonly used statements could in this way be conveniently re-used. The Control Unit has many fields in the microword that are never changed. All of these are gathered in the file "CU.fields".

The 'SIZE' statement informs the MASS assembler about the width of the microwords in bits.

Currently, MASS runs on SUN SPARC stations under the UNIX operating system.

Example. The following example shows how MASS is used to write a microroutine which adds two vectors A and B and puts the result back in vector A. The start addresses (pointing to the least significant bitslice) of the two vectors are stored in Register File registers R0 and R1 respectively. The fieldlength (number of bits per vectorelement) is stored in register R2. The PE structural design used in this example can be seen in Figure 6.

```
#SIZE 96

#INCLUDE "CU.fields"

#FIELD PE_INMUX 64 3 { IN_ZERO, IN_ONE, IN_CARRY, IN_UP, IN_SH, IN_BR, IN_MEM, IN_DOWN }

#FIELD PE_FUNCMUX 67 3 { R_R, R_AND, R_OR, R_XOR, R_ADD, R_XOR_T, R_MULT, R_IN }

#FIELD PE_CARRYMUX 70 2 { C_C, C_ADD, C_IN, C_XX }

#FIELD PE_TAGOPS 72 2 { TTT, TFT, *TTU, TFU }

#FIELD PE_MULT 74 3 {M_HLT, M_SH, M_MULT, M_MULT_SH, M_CLR, M_SH_CLR, M_XX, M_YY }
```

```
#FIELD PE_RW 77 1 { WRITE, *READ }

/* Decrement address to A. Clear Carry-bits and R-bits in all PE's */

 A_ADDR= R0, APMNEM = DECR_A, W_ADDR = R0, W_ENABLE = ENABLE, PE_INMUX= IN_ZERO, PE_FUNCMUX=R_IN, PE_CARRYMUX=C_IN;

/* Loop for number of bits (transfer R2 to counter register in sequencer) */

/* Address vector B */

SEQMNEM = FOR_D, DBUSSEL = REGFILE, A_ADDR = R2, B_ADDR = R1, APMNEM = ZERO_EXTB;


/* Increment address to vector A, address vector A*/

/* Move B-bit to the R flip/flop */

A_ADDR=R0, APMNEM = INCR_A, W_ADDR = R0, W_ENABLE = ENABLE, PE_INMUX= IN_MEM, PE_FUNCMUX=R_IN;


/* Address vector A */

/* Add one bit with carry */

A_ADDR = R0, APMNEM = ZERO_EXTA, PE_INMUX=IN_MEM, PE_FUNCMUX=R_ADD, PE_CARRYMUX=C_ADD;


/* Store result in vector A */

/* Increment address to vector B, address vector B */

/*Jump back in the loop if counter register <> 1 */

A_ADDR = R1, APMNEM = INCR_A, W_ADDR = R1, W_ENABLE = ENABLE, SEQMNEM = DJMP_S, MEM_CNTRL = WRITE;


/* Done, signal to the master */

STATDATA = DONE;

STATCLOCK = HIGH, STATDATA = DONE;
```

Soon there will also be a MACRO facility in MASS. This will greatly improve the readability of the source code, and will make the writing of MASS code easier.

5.2 High level micro-programming

Following the ideas of Tanenbaum [Tanenbaum, 1990] and the Lucas Microprogramming language [Fernström et.al, 1986] we intend to develop a structured microcode assembly language (SMAL) for REMAP^3. Tanenbaum's microcode language is inspired by high-level languages like Pascal and is based on assignments and goto's. For our

purpose we will also add program flow control structures like looping, if-then-else, while-do, and support for higher-language interface by a module concept.

SMAL will also try to hide the pipelining which is visible in MASS. It will also hide the register assignments necessary when arguments are passed from the Master Processor to the Control Unit.

To exemplify, the add_vec function implemented in MASS above could be written as:

```
void add_vec(int len, PE_address A, PE_address B)
{
  C = 0;  /* Clear carry */
  loop(len);/* Loops over the field */
  {
    R = M[A];/* Get a bit from A */
    R = R + M[B++];/* Add a B bit
                    (with carry) */
    M[A++] = R;/* Store Result in A*/
  }
}
```

Here we have assumed that C and R are two of the flip-flops in the PEs, and M[address] is corresponds to a memory position in the PE array.

6 IMPLEMENTATION

The implementation of the computer prototype is divided into two steps. In the first step the *alpha* version is developed. This version consists of the Control Unit, the Master Processor and the LUPUS processor card. This version incorporates 12 PEs of the kind shown in Figure 6 and has been successfully tested.

The Control Unit is made of 65 integrated circuits, connected by wire-wrapping. It occupies one double-height extended Europe card with the dimensions 233,5 * 220 mm.

The Master Processor is a commercial product - the PEP-VSBC2 card containing the 68000 processor and 216 kByte of memory. It occupies one single-height extended Europe card.

The LUPUS processor card is made of 53 integrated circuits, connected by wire-wrapping. It occupies one double-height extended Europe card with the dimensions 233,5 * 220 mm.

These 3 cards are housed in a 19” rack with a double contact VME-bus backplane. A power supply unit and a fan is also used. Figure 9 below show the complete arrangement.

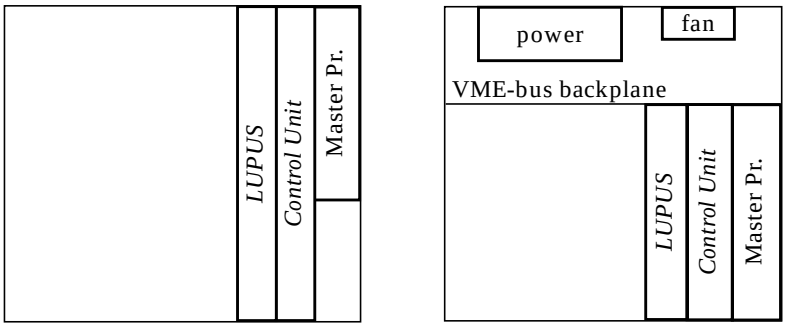

Figure 10 The 19" rack configuration – the alpha version

In the next step the *beta* version will be implemented. This will include the Control Unit (same as in the alpha version), the Master Processor and a new processor card. The processor card will be built with a new FPGA family, the XILINX X4000. This will result in a final implementation with at least 32 PEs per processor card, making the complete computer consist of at least 256 PEs.

In the beta version, the Master Processor will be extended by an Ethernet interface with the TCP / IPprotocol on-board.

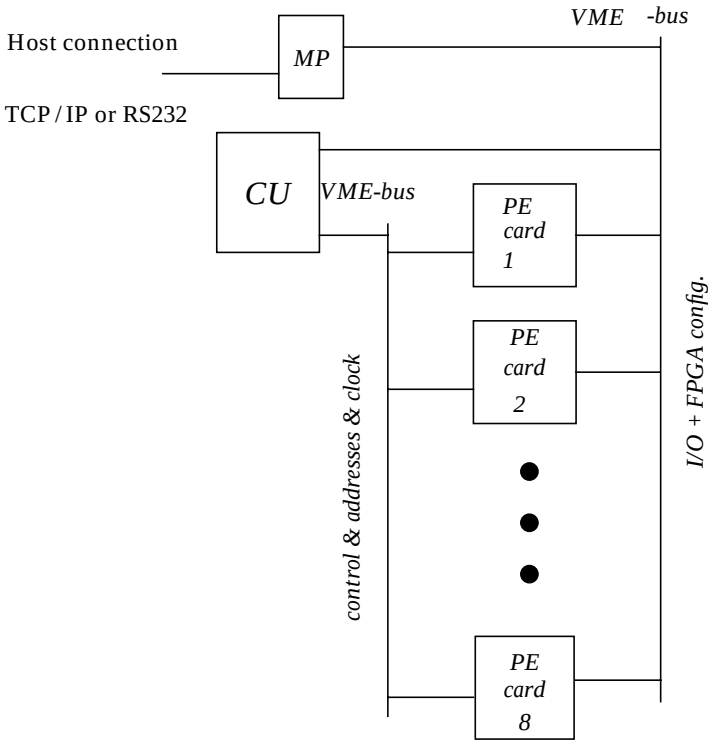

Figure 11 System overview, Beta version

7 APPLICATION SPECIFIC VARIATIONS

7.1 PE architectures for Neural Network Processing

The detailed studies of artificial neural network computations have resulted in a proposal for a PE that is well suited for this area. The design is depicted in Figure 9. Important features are the bit-serial multiplier and the broadcast connection. Notably, no other inter-PE connections are needed.

The PE is quite general purpose, and the hypothesis is that this is a useful PE design also in several other application areas.

As shown in [Nordström, 1991a] the incorporation of a counter instead of the multiplier in the PE design may pay off well when implementing the Sparse Distributed Memory (SDM) neural network model. A 256 PE REMAP^3 realization with counters is found to run SDM at a speed 10 - 30 times faster than that of an 8k PE Connection Machine CM-2 [Rogers, 1988]. Already without counters (then the PEs become extremely simple) a 256 PE REMAP^3 outperforms a 32 times larger CM-2 by a factor of 4 - 10. One explanation of this is the more developed Control Unit of REMAP^3.

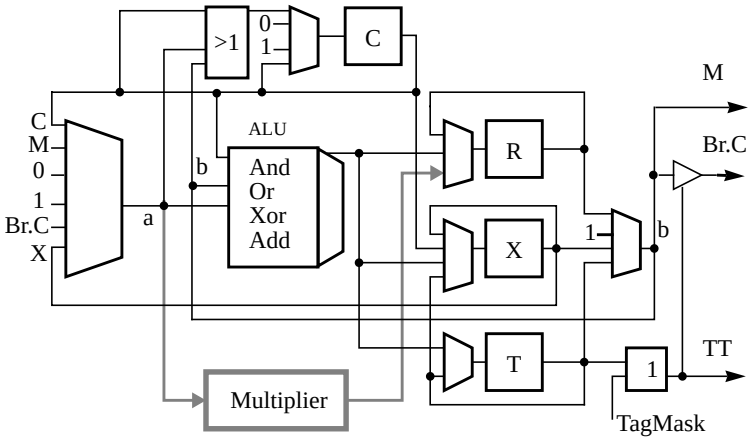

Figure 12 A PE for neural network processing

8 CONCLUSION

We have described the first steps in an experimental research project aiming at the design and implementation of highly parallel SIMD array modules from which embedded, massively parallel systems can be built. The early prototypes use programmable hardware to realize the PEs, and examples of PE designs implementable on the hardware have been given. A later step, which is now started, is the design of the inter-module communication. This should support the combination of two or more modules to one SIMD array as well as combinations of modules to form MIMSIMD (Multiple Instruction streams, Multiple SIMD arrays) architectures as shown in Figure 1. Flexibility of the I/O

systems to allow adaptation to the application data format is an important design criterion. The beta version will allow experimentation also in this respect.

Programming on the microprogramming level is important in the design and evaluation of the architectures. Different activities (debugging, instruction set development, algorithm develop-ment, etc.) require different levels of sophistication in the software tools, while still on the microprogramming level. The MASS and SMAL languages described illustrate this.

9 ACKNOWLEDGEMENT

We thank Michael Unnebäck who made a detailed study on the use of the new X4000 FPGA family in his Master's thesis project at Luleå University of Technology.

The REMAP^3 project is partly financed by NUTEK, the Swedish National Board for Technical Development, under contracts no 9001583 and 9001585.

10 REFERENCES

Bengtsson L. "A control unit for bit-serial SIMD processor arrays", Technical Report 9102, Centre for Computer Science, Halmstad University, Halmstad, Sweden.

Bengtsson L. "MASS - A low-level Microprogram ASSembler, specification", Technical Report 9103, Centre for Computer Science, Halmstad University, Halmstad, Sweden.

Bräunl, T. "Structured SIMD programming in Parallaxis." *Structured Programming*. Vol. 10(3): pp. 121-132, 1989.

Fernström, C., I. Kruzela and B. Svensson. *LUCAS Associative Array Processor – Design, Programming and Application Studies*, Vol. 216 of Lecture Notes in Computer Science, Springer Verlag, Berlin, 1986.

Linde A. and Taveniku M. "LUPUS – a reconfigurable prototype for a modular, massively parallel, SIMD computing system", Master's thesis 1991:028E, Division of Computer Engineering, Luleå University of Technology, January 1991. (In Swedish).

Nordström T. "Sparse distributed memory simulation on REMAP^3", Research Report No. TULEA 1991:16), Luleå University of Technology, Luleå, Sweden, 1991(a).

Nordström T. "Designing parallel computers for self organizing maps", Research Report No. TULEA 1991:17), Luleå University of Technology, Luleå, Sweden, 1991(b).

Nordström T. and Svensson B. "Using and designing massively parallel computers for artificial neural networks", To appear in *Journal of Parallel and Distributed Computing*, March 1992.

Rogers D. “Kanerva’s sparse distributed memory: an associative memory algorithm well-suited to the connection machine.” Technical Report No. 88.32, RIACS, NASA Ames Research Center, 1988.

Svensson B. “Parallel implementation of multi layer feedforward networks with supervised learning by back-propagation”, CDv Research Report 8902, Centre for Computer Science, Halmstad University, Halmstad, Sweden, June 1989.[1989a]

Svensson B. and Nordström T. “Execution of neural network algorithms on an array of bit-serial processors”, *Proc. of the 10th International Conference on Pattern Recognition: Computer Architectures for Vision and Pattern Recognition*, Atlantic City, New Jersey, June 16-21, 1990.

Tanenbaum, A. *Structured Computer Organization*, Third Edition, Prentice-Hall, 1990

TMC. “C* Programming Guide”, Version 6.0, Thinking Machines Corporation, 1990.

Treleaven, P. C. “PYGMALION neural network programming environment.” In *International Conference on Artificial Neural Networks*, Vol. 1, pp. 569-578, Helsinki, Finland, 1991.

Wiberg P. “Architectures for trainable mechatronical systems, and their programming”, Manuscript, Centre for Computer Science, Halmstad University, Halmstad, Sweden, 1991.