Lecture Notes in Computer Science 705, Herbert Grünbacher, Reiner W. Hartenstein Eds. Springer-Verlag
Field-Programmable Gate Array: Architectures and Tools for Rapid Prototyping; *Selected papers from: Second International Workshop on Field-Programmable Logic and Applications (FPL'92), Vienna, Austria*
pp.199-210

Using FPGAs to Implement a Reconfigurable Highly Parallel Computer

Arne Linde*, Tomas Nordström‡, and Mikael Taveniku*

* Department of Computer Engineering
Chalmers University of Technology,
S-41296 Göteborg, Sweden

‡ Division of Computer Engineering
Luleå University of Technology,
S-95187 Luleå, Sweden

E-mail: arne@ce.chalmers.se, tono@sm.luth.se, micke@ce.chalmers.se

Abstract. With the arrival of large Field Programmable Gate Arrays (FPGAs) it is possible to build an entire computer using only FPGA and memory. In this paper we share some experience from building a highly parallel computer using this concept. Even if today's FPGAs are of considerable size, each processor must be relatively simple if a highly parallel computer is to be constructed from them. Based on our experience of other parallel computers and thorough studies of the intended applications, we think it is possible to build very powerful and efficient computers using bit-serial processing elements with SIMD (Single Instruction stream, Multiple Data streams) control.
A major benefit of using FPGAs is the fact that different architectural variations can easily be tested and evaluated on real applications. In the primary application area, which is artificial neural networks, the gains of extensions like bit-serial multipliers or counters can quickly be found. A concrete implementation of a processor array, using Xilinx FPGAs, is described in this paper.
To get efficient usage and high performance with the FPGA circuits signal flow plays an important role. As the current implementation of the Xilinx EDA software does not support that design issue, the signal flow design has to be made by hand. The processing elements are simple and regular which makes it easy to implement them with the XACT Editor. This gives high performance, up to 40–50 MHz.

1 Introduction

The requirements for flexibility and adaptivity to different circumstances and environments have motivated research and development towards trainable systems rather than programmed ones. This is true especially for “action oriented systems” which interact with their environments by means of sophisticated sensors and actuators, often with a high degree of parallelism [2]. Response time requirements and the demand to accomplish the training task points to highly or massively parallel computer architectures.

In REMAP, the Real-Time, Embedded, Modular, Action-oriented, Parallel Processor Project [3], the potential of distributed SIMD (Single Instruction stream, Multiple Data streams) modules for realization of trainable systems is investigated. Each SIMD

module is a highly parallel computer with simple PEs tuned to efficiently compute artificial neural network algorithms.

Within the project, a series of studies have been performed [10–12, 16] concerning the execution of neural network 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 paradigm for artificial neural networks (ANNs) and that arrays of bit-serial PEs with simple inter-PE communication are surprisingly efficient. As multiplication is found to be the single most important operation in these computations, there is much to be gained in the bit-serial architecture if support for fast multiplication is added.

Using today's relatively large field-programmable gate arrays (FPGAs), it is possible to build an entire computer using only FPGAs and memory. Still, if a highly parallel computer is to be constructed out of them, each processor must be very simple. As shown in our studies of parallel computers for ANN, bit-serial PEs with SIMD control suit our computational needs, which makes it feasible to use FPGAs as a means to construct the first prototypes of our computers.

The computer built should not be seen as a final "product", it is more of an architecture laboratory, in which it is possible to change the architecture of each PE rapidly. Designing and compiling a new architecture takes about one week and downloading an already prepared architecture takes less than a second.

2 Applications

To realize action-oriented systems, the artificial neural network (ANN) models [6, 7] form a very important implementation class. As shown in [12] the demands on the architecture are quite moderate for standard ANN algorithms like feed-forward networks with back propagation, Hopfield networks, or Kohonen self-organizing maps. These models, like most of the ANN algorithms, use a very simple model of the neuron. Typically, an artificial neuron computes a weighted sum of its inputs, a nonlinear function is then usually applied to the sum, and the result is sent along to neighboring neurons, see Fig. 1. The power of ANN computations comes from the large number of neurons (nodes) and their rich interconnections via synapses (weights).

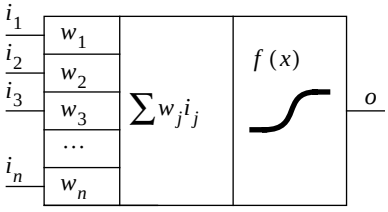

Fig. 1. The simplest model of a neuron. The neuron calculates the weighted sum of its inputs and applies a non-linear function to it, $o = f(\sum w_j i_j)$.

Different ANN models are characterized not only by the type of nodes, but also by the interconnection topology, and the training algorithm used [9]. Common topologies

are layered feed-forward networks, winner take all networks, and all-to-all (Hopfield) networks. Common training rules are error back-propagation and self-organizing feature maps.

Parallelism can be found in many different places [12] but for action-oriented systems the parallelism in the nodes and weights are the important ones (node and weight parallelism). As we are focusing on the ANN models in which one can count the number of nodes and weights in thousands, we will have a lot of parallelism available. These two types of parallelism also fit the SIMD concepts perfectly.

The calculation of the weighted sum is the most time consuming calculation and should therefore be supported architecturally by any computer intended for real-time ANN computations. Also the communication means between different ANN algorithms/modules as well as between these modules and the environment have to be carefully designed.

Another possible application area for the architecture we describe would be low-level image processing. As the architecture is not very different from architectures which are known to perform well on low-level image processing problems (e.g. AIS-5000 [14], LUCAS [5]), this problem area also fits our architecture well.

3 The REMAP Computer

REMAP is an experimental project. A sequence of gradually evolved prototypes are being built, starting with a small, software configurable PE array module, implemented as a Master's thesis project [8]. With only slight modifications in the PE array architecture, but using a new high-performance control unit, the second prototype has now been built[1]. This prototype is almost full-scale with respect to the number of PEs, but far from miniaturized enough for embedded systems. It is the architecture of this prototype that is described in this paper.

The computer consists of a number of computing modules controlled by a master computer. Each computing module is a SIMD computer of its own. It contains a linear array of bit-serial processing elements with memory and I/O-circuits controlled by a control unit, see Fig. 2.

1. A 128 PE prototype has now (beginning of 1993) been completed.

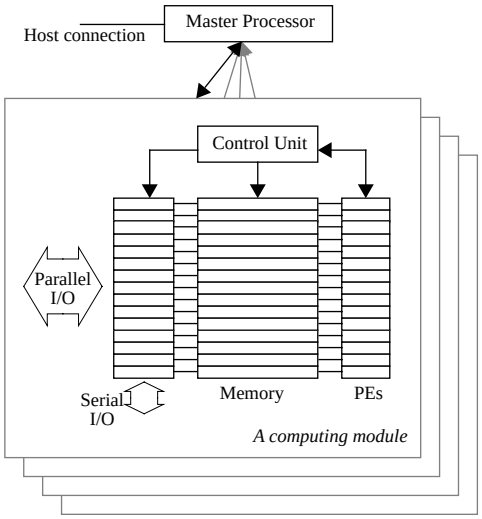

Fig. 2. Overview of the REMAP system. The PEs are implemented in Xilinx XC4005 circuits (8 in each) and the serial/parallel I/O device in Xilinx XC3020 (8 parallel and 8 serial I/O each)

3.1 The Control Unit

The main task for the control unit is to send instructions together with PE memory addresses to the PE array. At the same time it computes new address values (typically increments and decrements).

The control unit currently in use [3] has been designed around a microprogrammable sequencer and a 32 bit ALU (AMD 28331, 28332). The control unit is capable of sending out a new address together with a new instruction every 100 ns. The controller is more general purpose than usually needed, but until we know what is needed it serves our purpose. The microprograms to be executed by the control unit are stored in an 8K words control store. The operations can either be simple field operations, like adding two fields, or whole algorithms like an ANN computation. For the moment only a micro-code assembler is available to program the control unit, but we intend to develop more high level software development tools in the future. Currently we are looking into the possibility of using/developing a data-parallel language similar to C* [17].

3.2 PEs for ANN Algorithms

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 Fig. 3. Important features are the bit-serial multiplier and the broadcast connection. Notably, no other inter-PE connections than broadcast and nearest neighbor are needed. The PE is quite general purpose, and we are confident that this is a useful PE design also in several other application areas. In this version it consists of four flipflops (R, C, T and X),

eight multiplexers, some logic and a multiplication unit. The units get their control signals directly from the micro instruction word sent from the control unit.

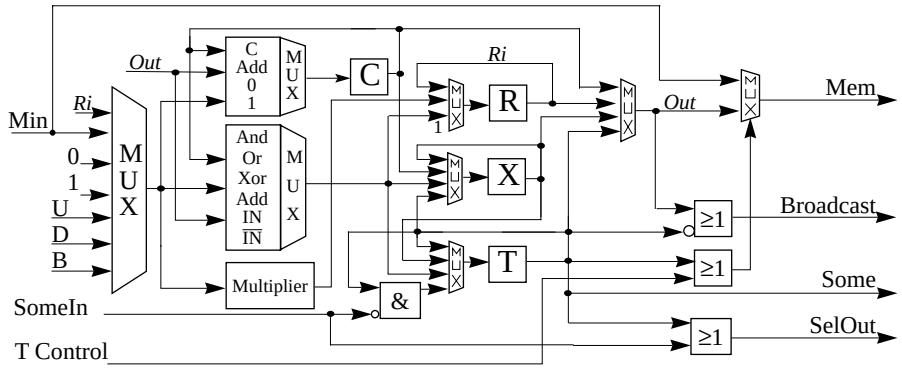

Fig. 3. The sample PE

In simple PEs without support for multiplication the multiplication time grows quadratically with the data length. A method based on carry-save adders [5] (see Fig. 4) can reduce the multiplication time required to the time to load the operands and store the result.

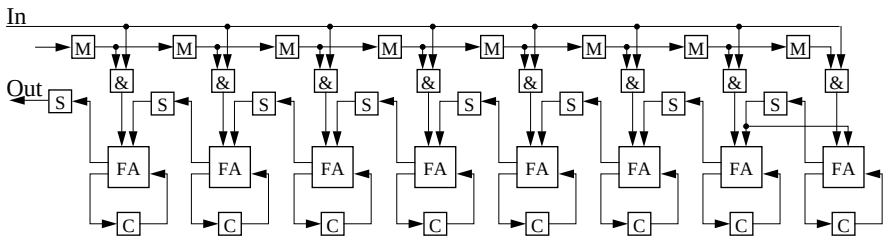

Fig. 4. Design of a two's-complement bit-serial multiplier. It is operated by first shifting in the multiplicand, most significant bit first, into the array of M flip-flops. The bits of the multiplier are then successively applied to the input, least significant bit first. The product bits appear at the output with least significant bit first.

As shown in [11] the incorporation of a counter instead of a multiplier in the PE design may pay off well when implementing the Sparse Distributed Memory (SDM) neural network model. A 256 PE REMAP realization with counters is found to run SDM at speeds 10–30 times that of an 8K PE Connection Machine CM-2, (with frequencies normalized and on an 8K problem). Already without counters (then the PEs become extremely simple) a 256 PE REMAP outperforms a 32 times larger CM-2 by a factor of 4–10. Even if this speed-up for REMAP can be partly explained by the more advanced sequencer, the possibility to tune the PEs for this application is equally important.

3.3 PE Communication

The processing element has two ways of communicating with other processing elements: nearest neighbor and broadcast communication. The nearest neighbor communication network allows each PE to read its neighbor's memory i.e. PE(n) can read from PE(n+1) and PE(n-1). The first and the last PEs are considered neighbors. At any time one of the PEs can broadcast a value to all other PEs or to the control unit. The control unit can also broadcast a value to the PEs. It has also a possibility to check if any of the PEs has the activity bit (T-flip-flop) set. If several PEs are active at the same time and the control unit wants one PE to broadcast, the control unit simply does a select-first operation, which selects the first active PE and deselects the rest. These communication and arbitration operations can be used to efficiently perform matrix computations as well as search and test operations sufficient for many application areas, especially artificial neural networks. To be useful in real-time applications which include interacting with a changing environment, high demands are put on the I/O-system. To meet these demands the processor array is equipped with two I/O-channels, one for 8-bit wide communication and the other for array-wide communications. This interface has a capability to run at speeds up to 80 MHz (burst) which, for a 256 PE array, implies a maximum transfer rate of 20 Gbit/s. Due to limitations in the control unit the I/O-interface currently runs at 10 MHz which reduces the transfer rate to 2.5 Gbit/s.

4 Designing with FPGA Circuits

After a market survey we found that FPGAs from Xilinx [22] would serve our needs best. The structure of the Xilinx circuits is shown in Fig. 5. The chip consists of a number of combinatorial logic blocks (CLB), some input-output blocks (IOB) and an interconnection network (ICN). These circuits are user programmable, thus enabling the CLB, IOB and ICN to be programmed by the user. The configuration of the on-chip configuration RAM is carried out at power up or by a reprogramming sequence. The RAM can be loaded from an external memory or from a microprocessor, the latter is used for REMAP. It takes about 400 ms to reprogram the circuits, thus enabling the master-processor to change the architecture of the processing elements dynamically during the execution of programs.

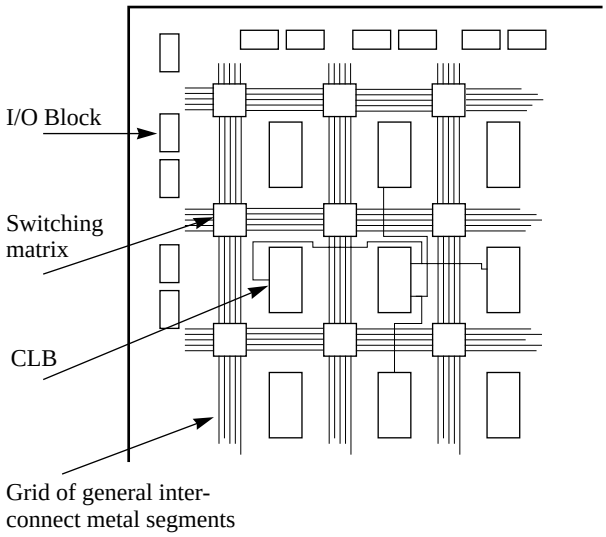

Fig. 5. Xilinx FPGA overview. The IOB connects the I/O-pads to the ICN. These blocks can be configured as input, output or bidirectional blocks. The CLBs are configurable logic blocks consisting of two 16bit (and one 8bit) look-up table for logic functions and two flipflops for state storage. These blocks are only connected to the ICN. The ICN connects the different blocks in the chip. It consist of four kinds of connections: short-range connections between neighboring blocks, medium-range connections connecting blocks on slightly larger distances, long-lines connecting whole rows and columns, and global nets for clock and reset signals broadcasted throughout the whole chip.

Since one of our goals is to make a kind of hardware simulator for different types of PE-architectures using a fixed hardware surrounding, it is required that the connections off chip like those to the memory and control unit have the same function regardless of the currently loaded processor architecture. As shown in [18] it is advantageous to lock the pads so that control signals enter from the top and bottom of the chip, and also design the processing elements so that they are laid out rowwise in the array of CLBs. It is likewise preferable to have a dataflow from left to right in the chip i.e. input data enters the left side and output emerges from the right side.

4.1 Using XC3090

The first prototype was constructed using Xilinx XC3090, and some frustrating experiences were gained from the poor development tools for these circuits. The processing elements in this version are only capable of running at 5 MHz clock frequency. The low speed is due to the incapability of the EDA software to handle signal flow layout in the circuits, something which also leads to low utilization. The PEs were designed using the OrCAD CAE-tools, enabling the designer to work with ordinary logic blocks like multiplexers and different types of gates. The schematics are then automatically converted to suit the Xilinx circuits. This is a fast design method but different parts of the logic become intermixed and long delays are introduced.

4.2 Using XC4005

The current prototype is based on the XC4000 FPGA family from Xilinx. These circuits have a more balanced performance than the XC3000 circuits which have small routing resources compared to the number of CLBs. In the XC4000 family the CLBs are larger, the ICN much more powerful and the internal delays shorter. The circuits range from XC4003, which has a 10 by 10 CLB matrix, to XC4010 with a 20 by 20 CLB matrix, and even larger circuits are announced. With these circuits it is easier to test new types of PEs, as there is more space in them. It will also be possible to increase the maximum clock frequency to 20 MHz, and possibly even 40 MHz if more pipelining is introduced. The greatest advantage with these new circuits is the software; routing a XC3090 chip can take a couple of days on a 80486 machine, while the same problem can be solved in half an hour with the new software for the XC4000 circuits.

One PE of the kind depicted in Fig. 3 occupies approximately 10 CLBs and the eight-bit deep bit-serial multiplier 11 CLBs. Using a XC4005 with a 14 by 14 CLB-matrix, we can get at least 8 PEs in each Xilinx chip. Considering this and the timing demands of 10 MHz operation (due to the control unit), we can easily make design variations both in the main processor and in the multiplier (or other coprocessor). It takes about one week to make a tested and simulated prototype with the XACT editor. The design is of course also open for changes to PEs with other data widths between 2 bits and eight bits.

Tools

High level tools were not available when we started to develop a processing element for the XC4005-circuits. Therefore we have not yet tested how well those tools work. There are, however, several advantages of using the low-level XACT editor in early stages of the design. We get good knowledge of the circuit's limitations and possibilities, and at the same time we get full control of all necessary timing. The usage of XACT is simplified by the regular and simple structure of our design. In the first implementation we aimed towards eight processing elements running at 10 MHz in each XC4005 circuit, based on the previous experiences with the XC3090 circuits. These goals were easily achieved; the eight processing elements can run at 20 MHz utilizing 75% of the XC4005 configurable logic blocks and all of its I/O blocks, this in the 84 pin PLCC package.

Data and Control Flow in the Circuits

The data and control flow play an important role in getting the best performance out of the circuits, therefore we have a basic template with some of the control and data signals already laid out. This template enables the user to easily implement new types of processing elements with minimum effort and at the same time achieve high performance.

When designing the control flow we want to use the global networks as much as possible. This is achieved by using 4 of the global nets and 20 of the vertical long-lines. The memory input signal is connected to the memory output via a horizontal long-line through the chip in order to enable a good data input distribution and allow

write-back of unchanged data when the processing element is inactive. With these restrictions in signal flow the internal delays can be held very low.

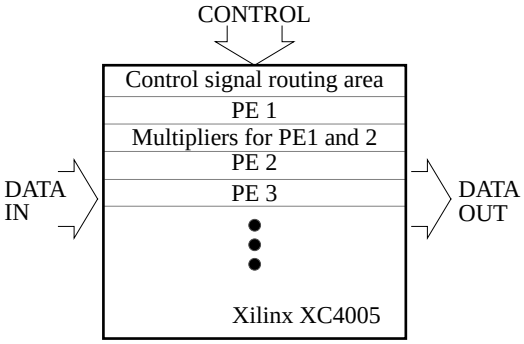

Fig. 6. Layout for PEs in an XC4005

As we have used the XC4005-PC84 which has a 14x14 CLB matrix, and the rest of the hardware is designed for eight processing elements, the chip is divided into four blocks of two processing elements each, occupying three rows in the matrix. Each processing element then gets 21 CLBs, 2 IOBs with PADs, and six IOBs with only edge decoders[1]. After this we have 28 unused CLBs.

Testability

From our experiences of the XC3090 circuits, which sometimes got into undefined states when we tried to reconfigure them, we now separate programming pins to the master processor so that we can directly see which circuit is failing. We also use the possibility of reading back configuration and state data from the Xilinx circuits, which can be done while the PEs are running. The master can also single-step the processor via the control unit and read back all state variables. Two pads on each processing element are dedicated to probing, here we can measure any internal delay simply by loading a configuration with the probe outputs properly programmed (this is done automatically by the XACT EDA software). The JTAG facilities of the XC4000 have not been used, because the PEs, simple as they are, only require a couple of hundred stimuli to excite all modes in them.

The full-scale prototype (256 PEs) can run in 10 MHz with very comfortable timing margins. More memory and additional communication networks can easily be added if need arises.

1. Some of the I/O-blocks in the XC4005-PC84 have no connections to pads. However, these blocks can be used to get a connection to their edge decoder.

5 Other Usage of FPGAs to Run ANN Algorithms

There are other FPGA implementations of ANN models besides ours. A short description of some of them are given below.

A group at North Carolina State University has developed a PC-card called Anyboard [19], which in principle only contains Xilinx chips (4 XC3020s) and RAM. It is part of a "rapid prototyping" environment, where user-specified digital designs can quickly be implemented and tested. One early project using this card was the implementation of a stochastic ANN model called TInMANN [20]. A quite fast and dense implementation was obtained. They used a special purpose architecture, tuned to their algorithm.

Another project, using 25 XC3020s to implement a stochastic Boltzmann machine ANN, was carried out by Skubiszewski [15]. In this implementation the architecture was more like ours (identifiable PEs similar to conventional bit-serial PEs), but no support for the multiplications was included.

Cox and Blanz [4] built an ANN simulator with impressive performance, out of 28 XC3090s. In contrast to the two implementations above and our implementation, they have used a highly specialized bit-parallel approach, which implements a feed-forward neural network of a fixed size (12x14x4).

Another, more specialized, use of FPGAs for ANN computations is made by a group at Tampere University of Technology, Finland [13]. In this group's hardware implementation of Kanerva's Sparse Distributed Memory (SDM), FPGAs are used to implement the main controller as well as more specialized computations like an adder tree. The architecture is highly specialized for SDM and no identifiable processing elements exist.

Xilinx circuits are also used in general hardware emulators such as the Quickturn RPM emulator [21], which emulates designs with from 10K up to 1M gates at a speed of 1MHz. This type of emulators could of course be used to simulate all the designs described in this paper, but with drastically lower speed and CLB utilization.

6 Conclusions and Future Directions

With the REMAP computer, we have a platform from where we can test and evaluate different types of interconnection networks, PE complexities and architectures. This is not restricted to simple bit-serial PEs as the one described in this text, also complex ones such as bit-serial floating point arithmetic units and up to eight bit wide PEs can be implemented. Floating point arithmetic for this platform has been examined by members of the group [1], and will be included. When we have found a good PE architecture we will transfer it to silicon, this decreases the size and increases the system speed. Our aim is to get 256 processing elements, with floating point arithmetic, on each chip running at an internal speed of 200–300MHz.

A robot arm with 12 motors and a number of sensors all controlled in parallel from the array-parallel interface on the REMAP computer is being developed at the Centre for Computer Architecture, Halmstad University. A CCD camera is also planned to be

connected to the byte-wide interface on REMAP as a further step towards a real-time action-oriented system.

To speed up the development cycle in the future some sort of high-level description of the PEs and their interconnections would be needed. From this description it should be possible to generate FPGA layout, VLSI layout, a PE array simulator, and a high level language compiler back-end. Both text-based and graphics based high-level descriptions are considered.

While, in this design of a hardware simulator, we are more interested in the possibilities of changing the processor architecture than to get maximum performance, we have added (retained) the feature that design changes can be made during execution. For example in some parts of an application we may need a counter instead of a multiplier. It is easily accomplished, via program control, to stop the control unit during approximately 400 ms and reprogram the Xilinx circuits.

7 References

1. Åhlander, A. and B. Svensson. "Floating point calculations in bit-serial SIMD computers." In *Fourth Swedish Workshop on Computer Systems Architecture*, Linköping, Sweden, 1992.
2. Arbib, M. A. "Schemas and neural network for sixth generation computing." *Journal of Parallel and Distributed Computing*. Vol. 6(2): pp. 185-216, 1989.
3. Bengtsson, L., A. Linde, T. Nordström, B. Svensson, M. Taveniku and A. Åhlander. "Design and implementation of the REMAP^3 software reconfigurable SIMD parallel computer." In *Fourth Swedish Workshop on Computer Systems Architecture*, Linköping, Sweden, 1992.
4. Cox, C. E. and W. E. Blanz. "GANGLION — A fast field programmable gate array implementation of a connectionist classfier." (RJ 8290 /75651/), IBM Research Division, Almaden Research Centre, 1990.
5. 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.
6. Hertz, J., A. Krogh and R. G. Palmer. *Introduction to the Theory of Neural Computations*. Addison Wesley. Redwood City, CA. 1991.
7. Kohonen, T. "An introduction to neural computing." *Neural Networks*. Vol. 1: pp. 3-16, 1988.
8. Linde, A. and M. Taveniku. "LUPUS — a reconfigurable prototype for a modular massively parallel SIMD computing system." (Masters Thesis 1991:028 E), University of Luleå, Sweden, 1991. [In Swedish]
9. Lippmann, R. P. "An Introduction to Computing with Neural Nets." *IEEE Acoustics, Speech, and Signal Processing Magazine*. Vol. 4(April): pp. 4-22, 1987.
10. Nordström, T. "Designing parallel computers for self organizing maps." (Res. Rep. TULEA 1991:17), Luleå University of Technology, Sweden, 1991.
11. Nordström, T. "Sparse distributed memory simulation on REMAP3." (Res. Rep. TULEA 1991:16), Luleå University of Technology, Sweden, 1991.

12. Nordström, T. and B. Svensson. "Using and designing massively parallel computers for artificial neural networks." *Journal of Parallel and Distributed Computing*. Vol. 14(3): pp. 260-285, 1992.

13. Saarinen, J., M. Lindell, P. Kotilainen, J. Tomberg, P. Kanerva and K. Kaski. "Highly parallel hardware implementation of sparse distributed memory." In *International Conference on Artificial Neural Networks*, Vol. 1, pp. 673-678, Helsinki, Finland, 1991.

14. Schmitt, R. S. and S. S. Wilson. "The AIS-5000 parallel processor." *IEEE Transaction on Pattern Analysis and Machine Intelligence*. Vol. 10(3): pp. 320-330, 1988.

15. Skubiszewski, M. "A hardware emulator for binary neural networks." In *International Neural Network Conference*, Vol. 2, pp. 555-558, Paris, 1990.

16. Svensson, B. and T. Nordström. "Execution of neural network algorithms on an array of bit-serial processors." In *10th International Conference on Pattern Recognition, Computer Architectures for Vision and Pattern Recognition*, Vol. II, pp. 501-505, Atlantic City, New Jersey, USA, 1990.

17. Thinking Machines Corporation. "C* User's guide and C* Programming Guide." (Version 6.0), T M C Cambridge, Massachusetts, 1990.

18. Unnebäck, M. "Gate array implementations of processing elements for a reconfigurable, modular, massively parallel SIMD computer." (Masters Thesis 1991:117 E), Luleå University of Technology, 1991. [In Swedish]

19. Van den Bout, D. E., J. N. Morris, D. Thomae, S. Labrozzi, S. Wingo and D. Hallman. "AnyBoard: An FPGA-based, reconfigurable system." *IEEE Design & Test of Computers*. (September): pp. 21-30, 1992.

20. Van den Bout, D. E., W. Snyder and T. K. Miller III. "Rapid prototyping for neural networks." *Advanced Neural Computers*. Eckmiller ed. North-Holland. Amsterdam. 1990.

21. Wolff, H. "How Quickturn is filling a gap." *Electronics*. (April): 1990.

22. XILINX. *The Programmable Gate Array Data Book*. 1990.