[5] Motorola MC68040 Microprocessors User's Manual, MOTOROLA INC. (1992).

that they cooperate in a practical manner.

7.4 nAPL; Message Layer

The lowest level of nAPL consists of an object–oriented interface to the coprocessor. There are two types of messages which can be sent from the host to the coprocessor:

1. Remote procedure calls (RPCs). These are "blocking" messages, i.e., the host waits until the procedure to be executed by the coprocessor is completed. RPCs are used to control operations which cannot be executed in parallel and, moreover, require only a short processing time.

2. Job Calls. These are "non–blocking" messages, i.e., when the corresponding C++ method is called, an order is issued to a task on the coprocessor to start a job, but the host does not wait for the end of the job. In this way it is possible to start a number of parallel running jobs one after the other. It is normal for several jobs to work simultaneously on processing a nAPL instruction. The coprocessor's operating system is responsible for control and administration of these jobs. On the higher software levels the host can make use of this to prepare the next jobs during that time.

Beneath the software pyramid as described above there are additional layers. They are related to the coprocessor's operating system and the hardware implementation of the jobs by means of microprograms. This software can be classified as firmware.

All of the software layers described here are open to users for the development of programs for SYNAPSE–1. The demands made of the users at each level are summarized below:

SENN++ simulator:	No programming required.
SENN++ class library:	C++ programmer. No knowledge of the hardware architecture necessary.
nAPL matrix level:	C++ programmer. Knowledge of the multiprocessor and memory architecture necessary.
nAPL message level:	C++ programmer. Knowledge of the multiprocessor and memory architecture and of the coprocessor operating system necessary.

The performance that can be attained with the programs increases from the top downwards. For real–time applications and compute intensive simulations, therefore, it is advisable to use nAPL.

8 Conclusions

The Neurocomputer SYNAPSE–1 is a very powerful and flexible device. Its size is adaptable to the applicational needs in terms of processing power and memory size. The time critical operations of neural network algorithms, e.g., the calculation of matrix–vector and matrix–matrix products, are accelerated by a systolic array of special purpose neural signal processors MA16. Depending on the algorithm used less time critical operations of neural network models are calculated by a multiple pipelined *Arithmetic Units* or by a multiprocessor system of general purpose microprocessors.

The system can be operated with a frequency of up to 40 MHz. The prototype system SYNAPSE–1 operates with computation rates of $5 \cdot 10^9$ connections per second and a weight storage capacity of 128 MBytes, and is therefore well suited for the VISION regions of the DARPA study [?]. One prototype system is operating since autumn 1992, other two systems are under test.

References

[1] DARPA Neural Network Study, AFCEA International Press, Fairfax, VA (1988).

[2] U. Ramacher, J. Beichter, N. Brühls, E. Sicheneder: Architecture and VLSI Design of a Neural Signal Processor, Proc. IEEE Int'l Symp. on Circuits and Systems, Chicago, IL, Vol. 3 (1993) 1975–1978.

[3] U. Ramacher, SYNAPSE—A Neurocomputer That Synthesizes Neural Algorithms on a Parallel Systolic Engine, J. Parallel and Distrib. Comp., Vol. 14 (1992) 306–318.

[4] U. Ramacher, W. Raab, J. Anlauf, U. Hachmann, J. Beichter, N. Brüls, M. Weßeling, E. Sicheneder, R. Männer, J. Gläß, A. Wurz: Multiprocessor and Memory Architecture of the Neurocomputer SYNAPSE–1; Proc. World Congr. on Neural Networks, Portland, OR (1993), Vol. IV (1993) 775–778

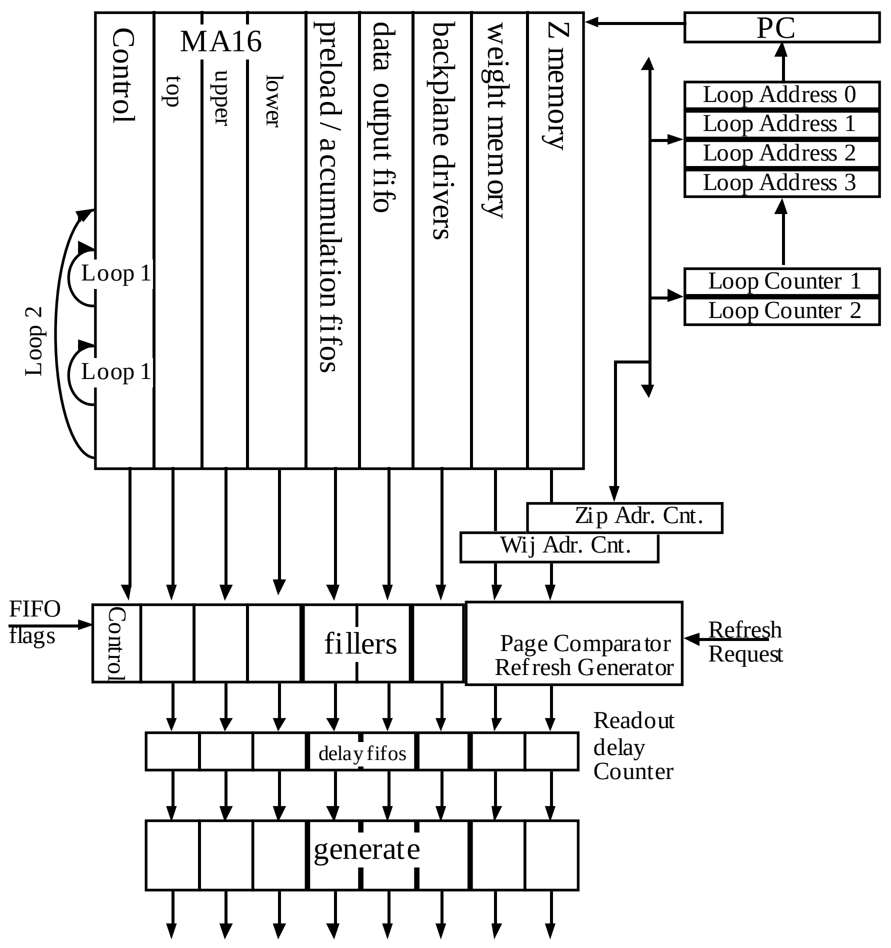

Figure 6: Programmable Sequencer.

The SENN++ class library is based on a package of subroutines containing an extensive range of operations defined on the basis of matrices and vectors. The operations are in the category of compute–intensive parallel operations, and are therefore closely linked with SYNAPSE–1 and the nAPL language.

7.3 nAPL; Matrix Layer

In its topmost layer, the nAPL C++ class library provides a simple interface for C++ programmers. Thanks to the use of the special hardware, programmers have new data types at their disposal: block–floating–point matrices and look–up tables (vectors are treated as special cases of matrices). A block–floating–point matrix comprises one 16–bit mantissa per matrix element and an exponent for the matrix as a whole. Look–up tables constitute the high–speed and universal hardware implementation of function evaluations in SYNAPSE–1. Programmers can access predefined function tables, or define new ones themselves.

There is a separate matrix class for each type of memory, Y, W and Z, each containing the elementary operations that can be executed for that memory. Thanks to the generous SYNAPSE–1 hardware it is possible to execute concatenations of compute–intensive and non–compute–intensive operations in one step. For example, an entire layer of neurons can be processed with a single nAPL instruction containing all parameters for the operations involved. This means that extremely powerful hardware–optimized instructions are available for execution.

Each of these instructions is executed with the aid of messages which initiate the parallel processes (tasks) on the *Control Unit* and *Data Unit*, ensuring trol word is read out and used to generate the control signals for these units. For more compact program coding two nested hardware loops are available with loop counters and address registers. An additional repeat function allows to repeat one instruction for a programmable number of clock cycles.

To ease programming and to avoid programming errors the sequencer words are not used directly for controlling the hardware. Rather additional hardware automatically inserts opcodes for refreshs of the dynamic memories into the instruction sequence from the program memory. During refreshs opcodes for "no operation" are generated to the other units like the MA16 array.

The program code for the sequencer is generated using a C++ cross compiler and a special assembler. Usual programs, like for standard matrix–vector products, are already compiled and can be loaded directly from a library. After some modifications the sequencer needs only to be started. Such modifications are, e.g., to set the start address for the weights and the loop counters. Due to the capacity of the SRAM of 256 k control words, several programs can be stored at the same time.

7 Software Architecture

The programming language for SYNAPSE–1 is based on the decomposition of each algorithm according to compute–intensive and non–compute–intensive operations. Since this decomposition equally determines the processor and memory architecture of SYNAPSE–1, the syntax of the programming language is a high level language mapping of the underlying hardware architecture of SYNAPSE–1. The declaration of a variable or a function is therefore accompanied by the specification of the memory or processor to which it is assigned. The programming language for SYNAPSE–1 is called nAPL (neural Algorithms Programming Language). It is embedded in C++ and is implemented as a class library. The power of nAPL is based on an extensive matrix library, which also contains complex operations (such as the complete processing of an entire layer of neurons, including transfer functions, with only one instruction). SYNAPSE–1 executes these nAPL operations at optimum performance. Its orientation with a matrix model also makes nAPL interesting for non–neural applications, such as general signal preprocessing.

Additional software layers (see Fig. ??) are required in order on one hand to establish the connection between nAPL and the procedures running on the *Control* and *Data Units*, and on the other hand to give users the option of non–hardware–dependent programming. The top two layers are entirely independent of the hardware, while the two layers beneath them are increasingly hardware-dependent.

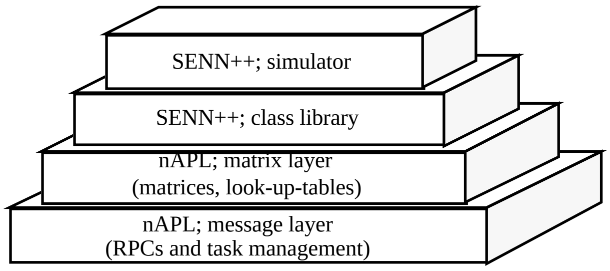

Figure 7: Software architecture of SYNAPSE–1.

7.1 SENN++; Simulator

The top level is occupied by the simulator component of SENN++ (Simulation Environment for Neural Networks). This is where the topology of a neural network, the algorithms to be used, and the data to be learned are entered, with the aid of an easy–to–use description language. An interactive graphical user interface allows users to monitor the simulation, change parameters, select data records for learning, and so on. Progress of the simulation can be illustrated on–line with a variety of graphics features.

At this level users are dependent on the existing objects (neurons, synapses, algorithms, etc.), and do not have to program anything nor have any knowledge of SYNAPSE–1. The option of programming by the user is provided on the next lower level.

7.2 SENN++; Class Library

The simulator is implemented with a class library which provides users with objects such as neurons, synapses, clusters of neurons, connectors between clusters, various methods of visualization, complete learning algorithms, administration of learning patterns, etc. A C++ programmer can use these classes for his or her own programs, derive new classes from the existing ones, and add new properties. This makes it possible to develop new algorithms with a minimum of programming effort. Provided that the programmer adheres to certain standards, the new classes automatically become available in the simulator and can be integrated there and manipulated interactively.

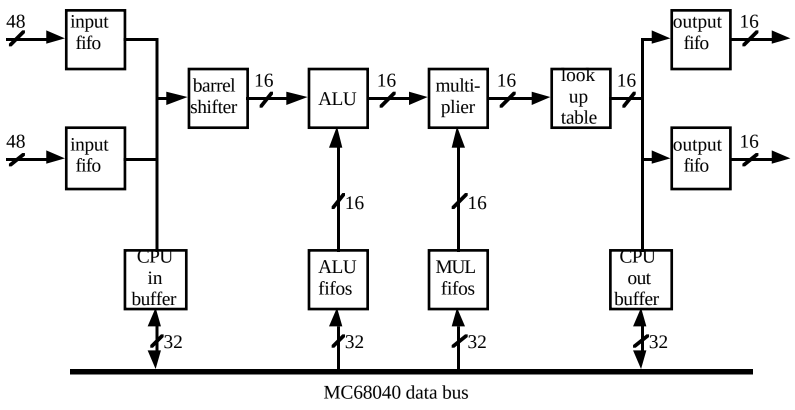

Figure 4: Arithmetic Unit.

namely identity and the Θ–function (a step function), are hardwired. Another arbitrary function can be programmed by the CPU. For more complicated algorithms for which the special hardware setup of the *Arithmetic Unit* is not suited, the *Arithmetic Unit* can be bypassed and the computation is executed by the CPU.

6 Control Unit

The *Control Unit* has two main responsibilities. First it coordinates all activities of SYNAPSE–1. This is done by a programmable sequencer that generates appropriate control signals and addresses for the other units. Second it provides the communication of the neurocomputer with a host system, usually a workstation used for programming and as the user interface.

On the *Contol Unit* there is also a Motorola CPU MC68040 with local memory and a VME interface, similar to the *Data Unit*. This CPU is used for the communication with the host and the CPUs of the *Data Units*, and for data transfer (Fig. ??). An *Address Generator* supplies the addresses used by the DMA controller for data transfer from the *Y–Memory* on the *Data Unit* to the *Neuroprocessor Units*. This *Address Generator* provides two programmable nested loops. Also single addresses generated by the CPU of the *Control Unit* can be processed.

The synchronous parts of the system like the MA16 array, the *Weigth Memory Unit* and the FIFOs, which connect the asynchronous *Y–Memory* or the *Arithmetic Unit*, are controlled by a programmable sequencer. Programming and control of the sequencer is

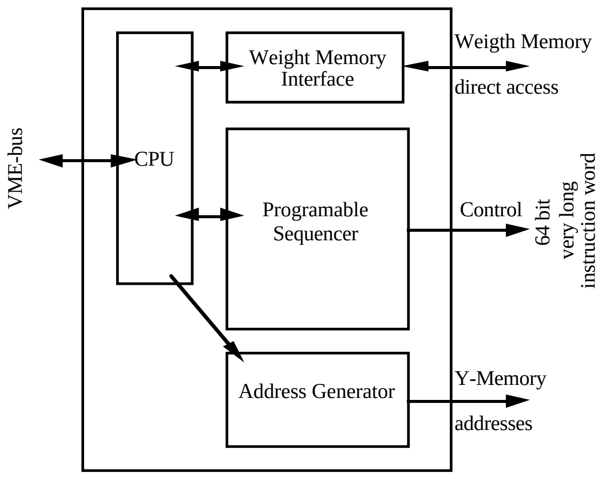

Figure 5: Control Unit.

done by the CPU. Partly integrated in this sequencer is the interface to the *Weight Memory Unit*. This allows to access the weight memory directly by the CPU.

6.1 Programmable Sequencer

The main part of the programmable sequencer is a SRAM in which programs are stored that consist of very–long–instruction–words (VLIW) of 64 bits width. The VLIW is divided into individual fields that are interpreted as coded commands for the units controlled by the sequencer. These are (see Fig. ??) the top, upper, and lower row of MA16 chips; the accumulation and data output FIFOs, the backplane drivers; and the weight and Z–memories. Every step one confloating point unit, a memory management unit, and cache memories for instructions and data. The CPU is currently operated at 33 MHz. The next version is planned to operate at 40 MHz. To provide high flexibility the processor allows I/O via two serial ports (RS232), and one 16–bit parallel port. For storing of monitor/debugger code and consolidated program modules 256 kBytes of EPROM are provided. The on–board I/O devices are able to generate interrupts of the MC68040 CPU. In addition some other conditions may also generate interrupt or bus error exceptions. These are overflow of the *Arithmetic Unit*, VMEbus slave interrupt, parity error in memory, and VMEbus error.

The memory consists of up to 32 MBytes DRAM and is used as instruction and data memory. It is multiported and can be accessed by the CPU and by external VMEbus masters via the VMEbus slave interface logic. Arbitration for memory access is normally performed cycle by cycle. In addition "locked" transfers are possible. Examples are RMW (read/modify/write) access by the MC68040 (TAS and CAS2 instructions), burst transfers for filling the on–chip cache memories, and RMW transfers and block transfers of the VMEbus slave logic.

Single memory accesses of the CPU can be executed at a rate of one transfer every 120 ns (33 MBytes/s). To perform burst mode transfers of the MC68040 in one clock cycle only, the memory architecture is two–fold interleaved. This allows a maximum transfer rate of 132 MBytes/s. Because three burst transfers require one "normal" transfer, the effective transfer rate is about 75 MBytes/s.

The VME master interface of the CPU supports "normal" and block transfers in extended address space. Fast read–out of data from another VME board, i.e., video boards, to the CPU or the *Y–Memory* of the *Data Unit* are handled by a dedicated VMEbus DMA (direct memory access) controller, that allows a transfer speed of up to 16 MBytes/s. This DMA controller operates in parallel with the CPU.

The VME slave interface is functionally completely independent of the master interface. It allows parallel access to the multiported data and instruction memory of the CPU, to the data memory of the *Neuroprocessor Unit*, and to control registers of the DMA controller and the *Arithmetic Unit*. To insure cache consistency of the on–chip caches of the MC68040 and the data and instruction memory it is planned to provide control logic that uses the snoop protocol of the CPU.

5.2 Y–Memory

In the *Y–Memory* the input data for the two rows of MA16 processors of the *Neuroprocessor Unit* are stored. After postprocessing by the *Arithmetic Unit* the results are also stored in this memory. For full–speed operation the *Y–Memory* consists of two banks of up to 16 MBytes each; one bank for read transfers and one bank for storing the results. The bank for storing results is simultaneously accessible via VME-bus. The different ports of each memory bank can be selected by a bank select register.

A DMA controller is provided that reads out data and transfers them to the *Neuroprocessor Units*. During learning a DMA controller writes back updated weights computed by the *Arithmetic Units*. For these accesses high speed FIFO buffers are used. With two–fold interleaved memory blocks and fast page mode access, transfer speeds of up to 100 MBytes/s are possible for reads and writes independently.

5.3 Arithmetic Unit

For postprocessing of the accumulated results of the matrix–vector products, as they are calculated by the *Neuroprocessor Units*, there is a special–purpose pipeline processor. In this *Arithmetic Unit* (Fig. ??) scaling factors and the transfer function of the neural algorithm are computed.

At the input of the *Arithmetic Unit* the data of the accumulation bus are stored in a FIFO buffer. Since these data have a width of 48 bits the most significant bits have to be selected for processing in the *Arithmetic Unit*. This is done by a barrel shifter that is controlled by the information provided by a max detector. During processing this max detector snoops on the accumulation bus of the *Neuroprocessor Unit* in order to detect and to store the position of the most significant bit.

The *Arithmetic Unit* consists of an ALU, a multiplier, and a look–up table. This unit is used alternately by both rows of neuro signal processors. ALU, multiplier, and look–up table operate in parallel in a pipelined structure. This allows to calculate one result in every clock cycle. Currently a clock of 33 MHz (max. 50 MHz) is used. Function parameters for the ALU and the multiplier are supplied by FIFOs, or by registers if constants are used. The data for these FIFOs and registers are supplied by the CPU. ALU and multiplier operation modes are programmable via control registers.

To calculate the transfer function, i.e., a threshold, a programmable look–up table is used. Two functions,

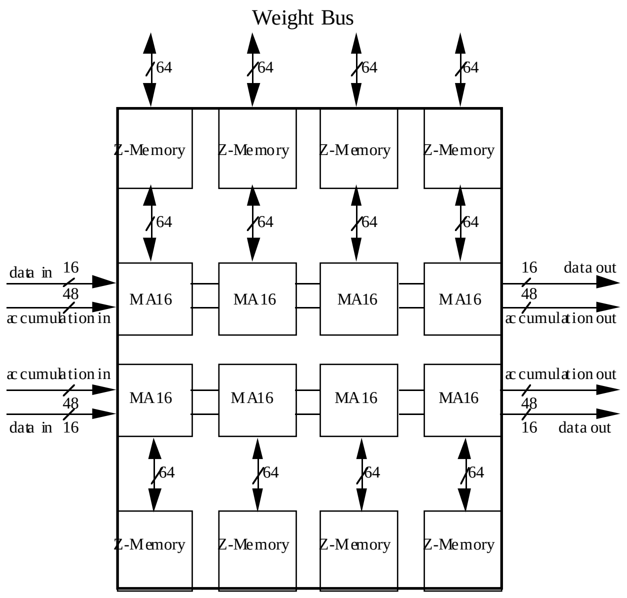

Figure 2: Neuroprocessor Unit.

16 × 16 bits fixed point integers whereas the results are summed up in 48 bits. With a 2×4 array of neuro signal processors MA16, a single *Neuroprocessor Unit* can compute up to $\approx 5 \cdot 10^9$ connections per second.

Each MA16 is accompanied by a local memory for intermediate data (Z–Memory). Both processors of a column are connected to the same weight bus. Thus they execute the same neural network operations for different input patterns. With a frequency of 40 MHz for the MA16 array and for the data buses, each MA16 has a bandwith of 320 MBytes/s on the weight bus and 80 MBytes/s at the data input. Since all processing elements operate in SIMD mode, i.e., execute the same operation on different data, programming of the systolic array is not difficult.

4 Weight Memory Unit

The *Weight Memory Unit* stores the synaptic weights. According to the 2×4 arrangement of the MA16 processors, the memory is divided into four identical parts. They are controlled such that data are provided synchronously to the systolic operation of the MA16 array. The four data streams are fed into the two MA16 processors that belong to the respective stage of the systolic chain. This is done via a special synchronous bus on the system backplane (Weight Bus) that transfers data at the system frequency of 40 MHz. With the bus width of 4×64 bits a total bandwidth of 1.28 GBytes/s is obtained. The total storage capacity of one unit is up to 512 MBytes. Because of the large storage capacity needed, DRAMs have been used for the memories. To reach the required high data rate, each memory module is two–fold interleaved and the DRAMs are operated in fast page mode.

For initialization of the weights, and for reading out and storing of learned weights, there is also an interface to the *Control Unit* for direct access. Via this connection the *Weight Memory* can be addressed sequentially using single and block transfers.

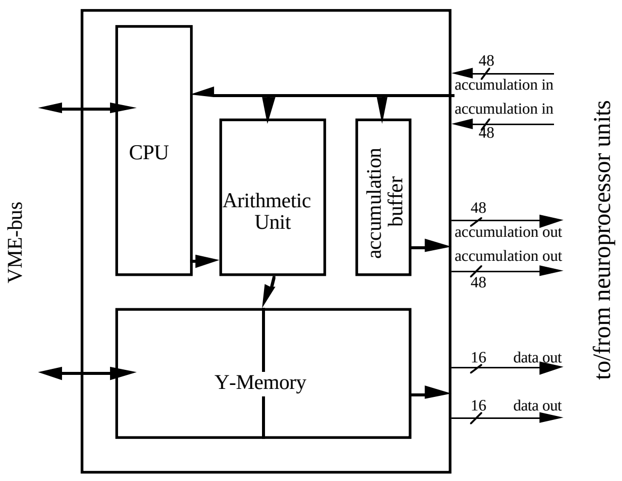

Figure 3: Data Unit.

5 Data Unit

A *Data Unit* feeds the data to be processed by the selected neural algorithm into the two rows of MA16 processors that are implemented on one *Neuroprocessor Unit*, receives their results, and postprocesses them (Fig. ??). The *Data Unit* is controlled by a Motorola MC68040 CPU with local memory and a VMEbus Interface. The data to be processed in the *Neuroprocessor Unit* are stored in a separate on–board memory (*Y–Memory*). In most cases postprocessing is done by a special purpose arithmetic unit tailored to the operations required in most neural algorithms.

5.1 CPU and VMEbus Interface

The processor module uses a Motorola MC68040 as CPU [?]. This 32–bit microprocessor has internally a

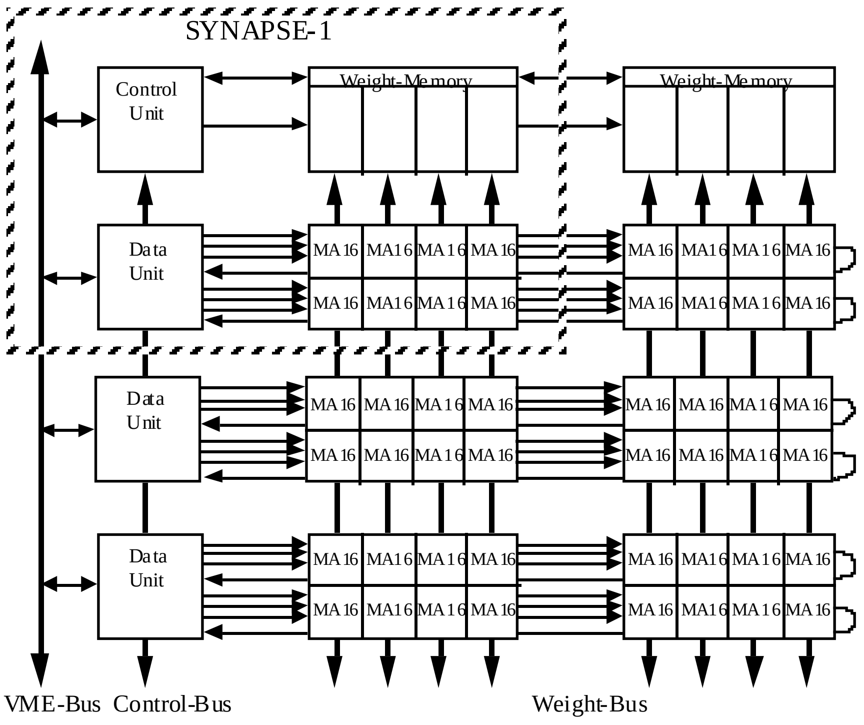

Figure 1: System architecture.

the results are routed back to the *Data Units.* Here all computational steps are executed that cannot be done in the processing array itself. Processing is done under central control by the *Control Unit.*

The processing power and the storage capacity of the memories can be adapted to the application needs. The minimal system configuration, which is operating since autumn 1992, consists of one *Neuroprocessor Unit*, one *Weight Memory Unit*, one *Data Unit*, and one *Control Unit* (see marked box in Fig. ??). For applications needing a higher input data rate the systolic array can be extended using more rows of *Data Units* and *Neuroprocessor Units.* In this way higher data parallelism is obtained by e distributing the same data over more processors operating with the same weights. For applications that require larger neural networks, i.e., more processing power as well as more weight memory space, the systolic array can be extended using more columns of *Weight Memory Units* and *Neuroprocessor Units.* System extension is easily achieved by using the appropriate number of boards in a system crate. The interconnection is provided by two special–purpose buses for weights and control signals, and a system bus. The VMEbus has been chosen as the system bus for simple interfacing to off–the–shelf hardware like the host computer or I/O devices.

3 Neuroprocessor Unit

The *Neuroprocessor Unit* executes the most time critical computations required for any simulation of neural networks including learning, e.g., the computation of matrix–vector products. It consists of a two–dimensional systolic array of neural signal processors MA16 [?], arranged in two rows by four columns (Fig. ??). For large matrix–vector products the MA16 processors calculate the results in several passes. In addition to the data and weight ports, each chip therefore provides input/output ports for partially accumulated results. These ports are horizontally connected between adjacent MA16 processors. In this way a row of MA16s form a linear systolic array where input data as well as partial results propagate from the *Data Unit* through the MA16s back to the *Data Unit.*

Each MA16 contains four systolic chains of four processing elements. A single MA16 processor, operating at 40 MHz, represents a computing power of $640 \cdot 10^6$ connections/s. Products are computed as

SYNAPSE–1: A High–Speed General Purpose Parallel Neurocomputer System

U. Ramacher, W. Raab, J. Anlauf, U. Hachmann
J. Beichter, N. Brüls, M. Weßeling, E. Sicheneder

ZFE ST SN43
Siemens AG
Otto–Hahn–Ring 6, D–81739 München, Germany

J. Gläß, A. Wurz
R. Männer

Lehrstuhl für Informatik V
Universität Mannheim
D–68131 Mannheim, Germany

Abstract

This paper describes the general purpose neurocomputer SYNAPSE–1 which has been developed in cooperation between Siemens Munich and the University of Mannheim. This system contains one of the most powerful processors available for neural algorithms, the neuro signal processor MA16. The prototype system executes a test algorithm 8,000 times as fast as a Sparc–2 workstation. This processing speed has been achieved by using a system architecture which is optimally adapted to the general structure of neural algorithms. It is a systolic array of MA16 processors embedded in a multiprocessor system of general purpose microprocessors.

1 Introduction

Neural algorithms are often well suited for industrial or medical image processing tasks. However it was not possible up to now to use such algorithms in real–time since the required computing power is much higher than provided by most computers [?]. SYNAPSE–1 is a neurocomputer that has new architecture which is optimally adapted to the general structure of neural algorithms. Its parallel architecture, a combination of a systolic array and a multiprocessor, allows SYNAPSE–1 to calculate artificial neural networks so fast that it will be possible, e.g., to process large images (up to 1024×1024 pixels) in real–time.

For the simulation of neural networks the most time critical operation is the computation of matrix–vector products. Since a new neuron state is calculated by multiplying the vector of the old neuron state by the synaptic matrix, such a product has to be calculated every iteration step. In many models of neural networks the number of iteration steps is comparable to the number of pixels in the image. Thus processing of even a single image is computationally demanding. If more than one image has to be processed as it is the case for real–time image processing as well as during the training phase of a neural network, matrix–matrix products have to be computed. Obviously only a highly parallel system is able to provide the required computation speed. In SYNAPSE–1 a systolic array of special neuro signal processors "MA16" [?] is used for this purpose. Each neuro signal processor contains 16 multipliers, 16 adders, and support logic like accumulators and FIFOs (first–in first–out memories).

For the calculation of the other parts of a neural algorithm, like scaling factors or the transfer function, the systolic matrix–vector calculation is embedded in an asynchronous multiprocessor system using Motorola's MC68040 CPUs. This allows to program the system easily in high level programming languages.

2 System Architecture

SYNAPSE–1 is a modular system whose building blocks are arranged in a two–dimensional structure [?, ?]. The building blocks are (Fig. ??) a two–dimensional array of neuro signal processors MA16, weight memories, data units, and a control unit.

The central part of the neurocomputer is the matrix of processing elements. The processing elements receive data from data units at the left edge of the processing array. The synaptic weights that are required for processing are input from the weight memories at the top edge. Partial results are computed and pipelined along the rows of the matrix to the right. After the matrix–vector products have been computed

REIHE INFORMATIK
11/95

SYNAPSE-1: A High-Speed General Purpose Parallel Neurocomputer System

J. Anlauf, J. Beichter, N. Brüls, J. Gläß,
U. Hachmann, R. Männer, W. Raab, U. Ramacher,
E. Sicheneder, M. Weßeling, A. Wurz
Universität Mannheim
Seminargebäude A5
D-68131 Mannheim