A General-Purpose Inference Processor for Real-Time Intelligent Controllers Using Systolic Arrays

C. Lucas* I.B. Turksen** K.C. Smith***

* Dept. of Electrical Engineering, Tehran University, Tehran, Iran

** Dept. of Industrial Engineering, University of Toronto, Toronto, Ontario, M5S 1A4, Canada

*** Dept. of Electrical Engineering, Computer Science, Library and Information Science, Mechanical Engineering, University of Toronto, Toronto, Ontario, M5S 1A4, Canada

Abstract

A systolic array implementation of a general-purpose inference processor is presented. The proposed processor can be used as a building block in the inference engine of an expert system or in a rule-based controller where computational speed is of importance.

1 Introduction

There has been increased interest, in recent years, in the design of intelligent controllers that model the behavior of human experts, in particular those who are capable of operating complex plants satisfactorily. This approach is most promising when the process or the plant that is to be controlled is so complex, with so little information available to the designer, that the conventional model-based approaches are not suitable. The functioning of an intelligent controller typically involves some sort of precise or approximate reasoning, since the controls to be applied are to be inferred from expert rules and the observations.

Although computers and software tools can be used to carry out the necessary computations and logical operations, in several real-time application, rapid on-line inference processing requires more efficient hardware implementations. On the one hand, the stringent real-time requirements call for designing special-purpose processors, while on the other hand, the extra design costs of special-purpose devices may reduce the economic efficiency of the proposed controller. A building-block approach can resolve the generality/performance problem. In the rest of this paper, a systolic architecture for real-time inference processing is presented. After a brief theoretical review of the approximate-reasoning routine called the Compositional Rule of Inference, a VLSI implementation exploiting the parallelism in that routine is presented. The paper concludes with a discussion of programmability and other problems related to the practical application of the proposed processors.

2 The Generalized Modus Ponens

Let **U** denote the plant input, to be determined by the controller from the measurement of the controlled output of the plant and its derivatives (or, equivalently, the estimation of the state of the plant), and the reference input, which can be denoted collectively by **X**. Expert knowledge can be modelled by production rules of the form

$$\begin{aligned} \mathbf{R_1} &: \textit{If } \mathbf{X} \textit{ is } \mathbf{A_1} \textit{ then } \mathbf{U} \textit{ is } \mathbf{B_1}. \\ \mathbf{R_2} &: \textit{If } \mathbf{X} \textit{ is } \mathbf{A_2} \textit{ then } \mathbf{U} \textit{ is } \mathbf{B_2}. \\ &\dots \\ &\dots \\ \mathbf{R_N} &: \textit{If } \mathbf{X} \textit{ is } \mathbf{A_N} \textit{ then } \mathbf{U} \textit{ is } \mathbf{B_N}. \end{aligned} \tag{1}$$

After observing that:

$$\mathbf{P} \ : \ \mathbf{X} \textit{ is } \mathbf{A}'. \tag{2}$$

it is sought to infer the value for **U**. Clearly, if $\mathbf{A}'$ coincides with $\mathbf{A_i}$, then one can "fire" the rule $\mathbf{R_i}$ and obtain the consequent $\mathbf{B_i}$. However, to allow for uncertainty and lack of precision, as well as incompleteness of the rule-set and possible non-coincidence of the observed premise with the antecedents of the rules, we model $\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_N}, \mathbf{B_1}, \mathbf{B_2}, ..., \mathbf{B_N}$, and $\mathbf{A}'$ as fuzzy subsets of the set of all possible values for **X** ($\{x_1, x_2, \dots, x_n\}$) and **U** ($\{u_1, u_2, \dots, u_m\}$). Thus they can be represented by n(m)-dimensional pattern vectors whose entries are membership functions taking values in the [0, 1] interval. Non-fuzzy (crisp) premises are special cases in which all the entries are either 0 or 1. Note that each entry in the crisp case represents one "bit" (binary unit) of information. So in the more general case, where entries take values in [0, 1] rather than only {0, 1}, each entry is said to represent one "fit" (fuzzy unit) of information.

The problem of deducing the value for **U**

$$\mathbf{Q} \ : \ \mathbf{U} \textit{ is } \mathbf{B}' \tag{3}$$

has been called the Generalized Modus Ponens and is also known as the Compositional Rule of Inference [1]. There are several approaches to the derivation of the conclusion $\mathbf{B}'$. In this paper, the following 3-step approach will be considered:

a. Representation of the rules $\mathbf{R_1}, \mathbf{R_2}, ..., \mathbf{R_N}$ as membership functions over the product space $\{x_1, x_2, ..., x_n\} * \{u_1, u_2, ..., u_m\}$;

b. Combination of the expert rules via the connective **AND (ALSO)** or **OR (ELSE)** [2,3];

c. Derivation of $\mathbf{B}'$ via the celebrated Compositional Rule of Inference.[1]

The first two steps can be carried out off-line, during a "learning" phase. The problem in the first step is that there are several different representations for the production rules suggested in the literature. A rather extensive listing of the different representations has been given in [2]. Basically, one uses the identity

$$\begin{aligned} \mathbf{R_i} &\equiv \{\mathit{if}\ \mathbf{X}\ \mathit{is}\ \mathbf{A_i}\ \mathit{then}\ \mathbf{U}\ \mathit{is}\ \mathbf{B_i}\} \\ &\equiv \{\mathbf{X}\ \mathit{is}\ \mathbf{A_i}\}^c \cup \{\mathbf{U}\ \mathit{is}\ \mathbf{B_i}\} \end{aligned} \tag{4}$$

where "^{c}" is the complementation operator.

The combination of the expert rules in the second step is carried out either via conjunction

$$\mathbf{R} = \mathbf{R_1} \cap \mathbf{R_2} \cap \ldots \cap \mathbf{R_N} \tag{5}$$

or via disjunction

$$\mathbf{R} = \mathbf{R_1} \cup \mathbf{R_2} \cup \ldots \cup \mathbf{R_N} \tag{5'}$$

The third step, however, has to be carried out online because the value for **U** has to be determined in real time for any observed value of **X**. In general, the conclusion $\mathbf{B}'$ is obtained via the Compositional Rule of Inference [7], for fixed j,

$$\mu_{\mathbf{B}'}(u_j) = \mathbf{S}_{x_i}(T(\mu_{\mathbf{A}'}(x_i), \mu_{\mathbf{R}}(x_i, u_j)) \tag{6}$$

where $\mu_{\mathbf{A}}(x)$ denotes the membership of x in **A**, and $\mathbf{T}(\bullet,\bullet)$ denotes the t-norm defined via

$$\mathbf{T}(\mu_{\mathbf{A}}(x), \mu_{\mathbf{B}}(x)) = \mu_{\mathbf{A}\cap\mathbf{B}}(x) \tag{7}$$

while $\mathbf{S}(\bullet,\bullet)$ is the corresponding co-norm

$$\mathbf{S}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}}) = 1 - \mathbf{T}(1-\mu_{\mathbf{A}}, 1-\mu_{\mathbf{B}}) \tag{8}$$

and $\mathbf{S}_{x_i}(\mu(x_i, u_j))$, $i = 1, \ldots, n$, denotes $(\mathbf{S}(\mu(x_1, u_j), \mu(x_2, u_j), ..., \mu(x_n, u_j))$ where

$$\begin{aligned} \mu(x_1, u_j) &= \mathbf{T}(\mu_{\mathbf{A}'}(x_1), \mu_{\mathbf{R}}(x_1, u_j)) \\ &\cdots \\ &\cdots \\ \mu(x_n, u_j) &= \mathbf{T}(\mu_{\mathbf{A}'}(x_n), \mu_{\mathbf{R}}(x_n, u_j)) \end{aligned}$$

(Note that both **T** and **S** are associative operators [5]). The choice of the t-norm and conorm is application-dependent. The most commonly used and theoretically justifiable operators are

$$\mathbf{T}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}} = \mathit{min}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}}) \tag{9}$$

$$\mathbf{S}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}} = \mathit{max}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}}) \tag{10}$$

Other well-known conjugate pairs are

$$\mathbf{T}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}}) = \mu_{\mathbf{A}} \bullet \mu_{\mathbf{B}} \tag{11}$$

$$\mathbf{S}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}}) = \mu_{\mathbf{A}} + \mu_{\mathbf{B}} - (\mu_{\mathbf{A}} \bullet \mu_{\mathbf{B}}) \tag{12}$$

and

$$\mathbf{T}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}}) = \mathit{max}(0, \mu_{\mathbf{A}} + \mu_{\mathbf{B}} - 1) \tag{13}$$

$$\mathbf{S}(\mu_{\mathbf{A}}, \mu_{\mathbf{B}}) = \mathit{min}(1, \mu_{\mathbf{A}} + \mu_{\mathbf{B}}) \tag{14}$$

Besides the numerous other conjugate pairs, one can also consider interval-valued representations based on conjunctive and disjunctive normal forms, which contain the above cases [6, 7].

3 A Systolic Architecture

The general-purpose inference processor that can be used as a building block in the design of intelligent controllers has the following characteristics:

a. A pattern vector representing the observed premise $\mathbf{A}'$, whose i-th component is the membership grade $\mu_{\mathbf{A}'}(x_i)$ of x_i in $\mathbf{A}'$, is fed to the processor as input;

b. A knowledge base, consisting of the expert rules (1), represented as a joint membership function via (4) and combined via (5) or (5′), is stored in the long- term memory of the processor;

c. The processor computes the possibly fuzzy conclusion $\mathbf{B}'$ via the Compositional Rule of Inference (6) [7] and produces the pattern vector for $\mathbf{B}'$, whose j-th component is the membership grade $\mu_{\mathbf{B}'}(u_j)$, as its output. In most applications, this output has to be defuzzified before being fed as the actuating signal to the plant.

In this paper, a systolic-array implementation, designed to take advantage of parallelism in the computation, and suited to very-large-scale integration (VLSI) and wafer-scale integration (WSI), is considered. The design procedure essentially follows [8].

One difficulty in extracting the parallelism in an algorithm arises when the algorithm is expressed in some conventional programming language which has a built-in sequential ordering of the computations involved, which ordering obscures any parallelism present in the algorithm. Also, the habit of overwriting of the variables in order to minimize memory requirements, left from the days when memory was to be used sparingly, further compounds the problem of extracting the parallelism of an algorithm. To avoid these difficulties, one can express the algorithm in the so-called Single-Assignment Language [8]. Every variable defined in the program takes on a unique value during the entire computation process. Next, a dependence graph can be drawn whose nodes represent the variables in the algorithm, and whose edges represent direct dependence so that a directed arc from node x to node y signifies that the value of x is used when computing y in the algorithm.

An important subclass of single-assignment algorithms, containing all algorithms executed by systolic arrays, are the so-called Regular Iterative Algorithms. In order to design a systolic architecture for a general-purpose inference processor via the Compositional Rule of Inference (6), the latter must first be expressed in recursive form

$$\mu^{(i)}_{\mathbf{B}'}(u_j) = \mathrm{S}(\mu^{(i-1)}_{\mathbf{B}}(u_j),\ \mathrm{T}(\mu_{\mathbf{A}'}(x_i), \mu_{\mathbf{R}}(x_i, u_j))) \quad (15)$$

with initial values

$$\mu^{(0)}_{\mathbf{B}'}(u_j) = 0. \quad (16)$$

The dependence graph of the algorithm can now be drawn in the index space (Fig. 1a). There is one node corresponding to each (i,j), (x_i, u_j) junction representing the computation of $\mu^{(i)}_{\mathbf{B}'}(u_j)$. The value for $\mu_{\mathbf{A}'}(x_i)$is supplied to the i-th column nodes, and the value for $\mu_{\mathbf{B}}(u_j)$ is computed at the n-th iteration (last column) of the j-th row. The values for $\{\mu_{\mathbf{R}}(x_i, u_j) : i = 1, 2, ..., n;\ j = 1, 2, ..., m\}$ are precomputed and stored at corresponding nodes.

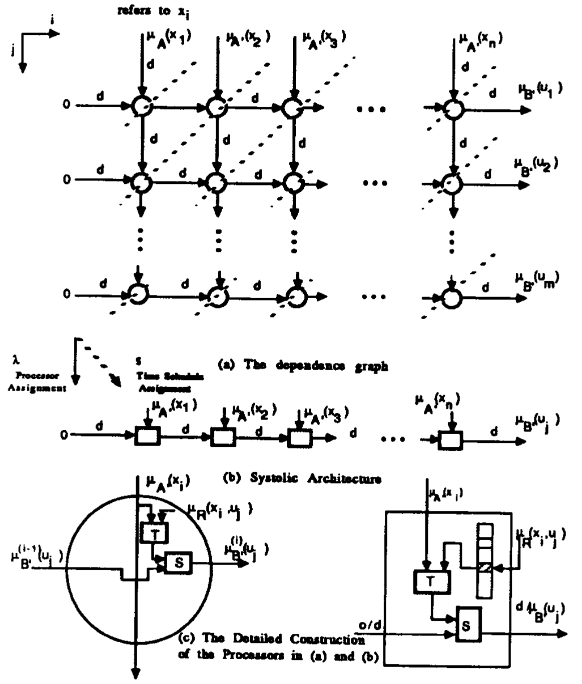


Figure 1: The Desgin Of A Systolic Array For Carrying Out The Compositional Rule Of Inference

The systolic-array architecture for this dependence graph can now be derived by specifying the processor and schedule assignment, i.e. by specifying which computation (represented by a node in the dependence graph) should be carried out by which processor and when. The constraints are: i) the set of nodes in the dependence graph assigned to the same processor should not be scheduled to be simultaneously computed; and ii) the schedule for computing the different nodes should be such that the antecedent nodes should already have been computed during the previous beat of the systolic clock.

If we denote the vector normal to the set of nodes to be computed by the same processor as λ (the processor assignment), and the vector normal to the set of nodes to be computed simultaneously as s (the time-schedule assignment), then from the constraints discussed above we derive the conditions

$$\lambda^{\mathrm{T}}\mathbf{s} \neq 0 \quad (17)$$

and

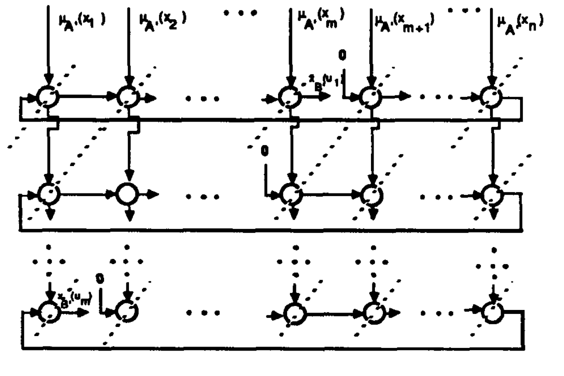
$$\mathbf{s}^{\mathrm{T}}\mathbf{d} > 0 \quad (18)$$

for any displacement vector **d** in the index space (any edge in the dependence graph). Now for the algorithm shown in Fig. 1a, the choice of $\mathrm{S} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, i.e. the time-schedule assignment shown via the dashed lines, clearly satisfies (18). To facilitate the provision of the input pattern vector, the processor assignment can be chosen so that $\lambda = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. And $\mu_{\mathbf{A}'}(x_i)$ is the input to the i-th processor during the entire computational process, so that (17) is also satisfied.

The detailed structure of the processor is shown in Fig. 1c. In the index space, each node stores the joint membership function $\mu_{\mathbf{R}}(x_i, u_j)$ representing the production rules from which the inference is to be made. During the i-th beat of the systolic clock, $\mu_{\mathbf{A}'}(x_i)$ and $\mu^{(i-1)}_{\mathbf{B}'}(u_j)$ are used for computing $\mu^{(i)}_{\mathbf{B}'}(u_j)$ via (15), which together with $\mu_{\mathbf{A}'}(x_i)$ is fed to the next set of nodes in the dependence graph for carrying out the computation at the next beat of the clock.

Since the computations in each column of the index space are to be carried out by the same processor, each processor must have an m-dimensional circular storage area where the values for $\mu_{\mathbf{R}}(x_i, u_j)$ for j=1,2,...,m are stored. These values are cyclically piped to the processor during the n+m beats of the systolic clock. The values for $\mu_{\mathbf{A}'}(x_i)$ are fed (and refed) to the processor from above, as well as initial values of 0 (zero) for $\mu^{(0)}_{\mathbf{B}'}(u_j)$ from the left. The first output appears in the right-hand-side output of the last processor after n beats of the clock, and during the next m beats of the clock one gets the entire output pattern $\{\mu_{\mathbf{B}'}(u_j), j = 1, 2, ..., m\}$ from the output.

Fig. 2 shows an improved version of this design. The order of computing (6) recursively as in (15) is permuted for the different values of j, and the schedule assignment is such that m processors are busy carrying out the computations corresponding to m nodes in the dependence graph at any beat of the clock (Fig. 2a). As a result, a ring systolic array as in Fig. 2b can carry out all the computations during n beats of the systolic clock.


(a) The Dependence Graph

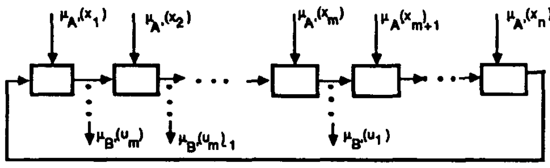

(b) The Systolic Architecture

Figure 2: An Improved Systolic Design For Compositional-Rule Infernece Processing

At the beginning of the process, the left-hand-side inputs of all processors are initialized to zero. The detailed structure of the processors is the same as before, and $\mu_{\mathbf{A}'}(x_i)$ is fed (and refed) to the i-th processor during the entire computational process. After n beats of the clock the output pattern $\{\mu_{\mathbf{B}'}(u_j), j = 1, 2, ..., m\}$ can be obtained from the right-hand-side outputs of the first m processors.

Care must be taken in both systolic realizations (Fig. 1 and 2) that the correct value for $\mu_{\mathbf{R}}(x_i, u_j)$ be fed to the t-norm computational unit in each processor. In the first realization, $\mu_{\mathbf{R}}(x_1, u_1)$ should be piped to the first processor at the first beat of the clock. At the second beat of the clock, $\mu_{\mathbf{R}}(x_1, u_2)$ is piped to the first processor while $\mu_{\mathbf{R}}(x_2, u_1)$ is piped to the second processor. At the i-th beat of the clock, $\mu_{\mathbf{R}}(x_i, u_1)$ is piped to the i-th processor, $\mu_{\mathbf{R}}(x_{i-1}, u_2)$ is piped to the (i-1)-th processor, and, ..., $\mu_{\mathbf{R}}(x_1, u_i)$ is piped to the first processor. Thus after each beat of the clock, $\mu_{\mathbf{R}}(x_i, u_k)$ is replaced by $\mu_{\mathbf{R}}(x_i, u_{k+i})$ at the i-th processor, and when k becomes equal to m, $\mu_{\mathbf{R}}(x_i, u_m)$ is replaced by $\mu_{\mathbf{R}}(x_i, u_1)$ at the next beat. The same cyclical piping of $\mu_{\mathbf{R}}(x_i, u_j)$ should occur in the second realization, but the values piped during the first beat of the clock should coincide with the values piped during the (m+1)-th beat of the clock in the first realization.

4 Implementation and Extensions

In general, the inference rule is expressed as an **S-T** composition. Different **S** and **T** operators can be chosen to fit specific applications. The most commonly used **S** operator is the maximum operator (10) while the most commonly used **T** operator is the minimum operator (9). These can be realized via circuits involving CMOS technology [9]. Other **T** and **S** operators, not necessarily conjugate pairs, can also be chosen for specific applications. In the current literature, while the **S** operator has almost always been the maximum operator (10), various **T** operators have been used [3, 10]. The product operator (11) is also easy to build, and so is the bounded product (13). The other commonly used **T** operator in compositional-rule inferring is the drastic-product operator

$$\mathrm{T}(a,b) = \begin{cases} min(a,b) & \text{if } max(a,b) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

Thus it is seen that the basic processors shown in Fig. 1c can be implemented for different choices of the **T** and **S** operators very easily via transistor circuits.

Another important consideration in the utilization of systolic-array inference processors in real-time control applications is the need to defuzzify the inferred actuating signal **U**, before feeding it to the plant. It has been assumed so far that the defuzzification process can be performed by the host processor. But so as to avoid the problems associated with off-chip defuzzification, especially since very quick inference processing is the goal, integrating the inference processor and a defuzzifier in one general-purpose intelligent controller unit may improve its performance as a building block to be utilized in different applications.

There are several different methods for defuzzification [10], the most commonly used being the centroid method

$$\mathbf{u} = \frac{\sum_{j=1}^{m} u_j \mu_{\mathbf{B}'}(u_j)}{\sum_{j=1}^{m} \mu_{\mathbf{B}'}(u_j)} \tag{20}$$

The defuzzification algorithm can easily be carried out recursively. It is, therefore, easy to perform the defuzzification within the general-purpose inference processor block by adding a defuzzifier unit to the right-hand end of the systolic architecture shown in Fig. 1b. After n beats of the systolic clock, the membership functions $\{\mu_{\mathbf{B}'}(u_j), j = 1, 2, ..., m\}$ will appear in the right -hand-side output and be fed to the defuzzifier unit during the next m beats of the clock. So the defuzzified value for **U** (denoted as **u**) will appear at the right-hand-side output of the defuzzifier unit after n+m beats of the clock. It is also possible to design the entire controller system so that it includes a library of fuzzified patterns corresponding to different linguistic variables describing the state X.

Another important design consideration is the programmability of the inference processor: Although the computation of $\mu_{\mathbf{R}}(\bullet,\bullet)$, representing the set of rules (1), can be carried out off-line, the inference processor would be of much more utility if those computations did not have to be carried out off-chip.

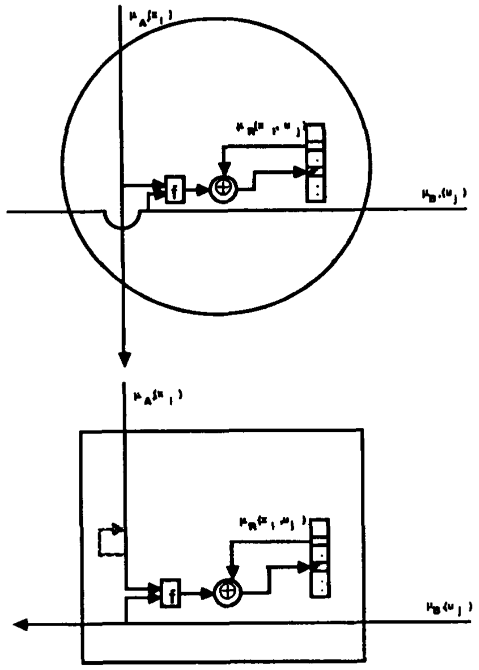


Figure 3: The Detailed Construction for the Learning Mode

It can be noted that the inference processor proposed in this paper is capable of learning new rules by changing the values for $\mu_{\mathbf{R}}(\bullet,\bullet)$ that are stored in its processing unit. Conveniently, this can be done recursively. Suppose we have the set of rules (1) represented via $\mu_{\mathbf{R}}(\bullet,\bullet)$ and we want to add a new rule. All we have to do is to represent the new rule via (4) (see also [10] for other representations), and then combine them with the existing rules via (5) or (5′) (or any other combination procedure [3]). Thus, the learning rule can be written as

$$\mu_{\mathbf{R}}(x_i,u_j) = \mu_{\mathbf{R}}(x_i,u_j) \oplus \mathrm{f}(\mu_{\mathbf{A}'}(x_i), \mu_{\mathbf{B}'}(u_j)) \quad (21)$$

where $\mathbf{A}'$ and $\mathbf{B}'$ are the antecedent and consequent of the new rule, $\oplus$ is the combination operator, and $f(\bullet,\bullet)$ is determined by the choice of the appropriate representation for the implication rule $\mathbf{A}' \rightarrow \mathbf{B}'$ [2].

The systolic architectures shown in Fig. 1 and 2 are capable of carrying out this in a learning mode of operation, in which the directions of data flow are reversed (and the patterns $\mu_{\mathbf{A}'}(\bullet)$ and $\mu_{\mathbf{B}'}(\bullet)$ are represented as inputs), and the cyclical piping direction of $\mu_{\mathbf{R}}(\bullet,\bullet)$ is also reversed, so that the initial configuration in the learning mode would correspond to the configuration at the final beat of the systolic clock for the inference- processing operation. The detailed construction in the learning mode is shown in Fig. 3. As in the previous case, the basic processors can be implemented for the different choices of the **f** and $\oplus$ operators via simple transistor circuits computing sums, products, maximums, minimums, etc.

5 Conclusions

A systolic architecture for inference processing has been proposed. The derivation of the architecture is based on the direct application of the Compositional Rule of Inference. The knowledge base consists of a set of production rules that can be represented via joint-membership functions. These representations are combined before the composition operator is applied, so that they can be stored within the inference processor in a more compact manner. There is no essential limit to the number of production rules in the knowledge base. There is also a sizable saving in the number of computations because one is not required to carry out a composition for each production rule.

An important feature of the proposed design is the learning capability of the inference processor. New production rules can be learned and stored inside the inference processor by combining them with the production rules constituting the current knowledge base. The same systolic architecture can be utilized for the learning mode of operation. This capability enables the inference processor to be "trained" for inferring conclusions from a different knowledge base. So, in particular, inference processors can be reprogrammed for controlling different plants or systems, or used adaptively for controlling systems whose behavior changes with time.

There has been remarkable progress in recent years in constructing VLSI inference processors. The first commercial products are now being introduced into the market. This paper has resulted from more general consideration of the development of new algorithms for inference processing, and the possibility of their implementations with systolic arrays and artificial neural networks. Our simulation results for a number of test problems conducted at the University of Toronto and Tehran University encourages us to believe that it is possible to build single-chip processors exploiting the parallelism inherent in systolic computation with impressive gains in power and efficiency.

References

1. Lofti A. Zadeh, "Knowledge Representation in Fuzzy Logic", IEEE Trans. Knowledge and Data Engineering, 1, 1, March 1989, 89-100

2. Zhigiang Cao and Abraham Kandel, "Applicability of Some Fuzzy Implication Operators", Fuzzy Sets and Systems, 31, 1989, 151-186

3. Meng-Hiot Lim and Yoshiyasu Takefuji, "Implementing Fuzzy Rule-Based Systems on Silicon Chips", IEEE Expert, 5, 1, Feb. 1990, 31-45

4. Caro Lucas, "On the Design and Implementation of a Class of Self-Organizing Intelligent Controllers", in Control and Modelling, C. Lucas and H. Ahmdi eds., Tehran University and IASTED Pub., Tehran, Iran, 1990, 254-257

5. B. Schweitzer and A. Sklar, "Associative Functions and Statistical Triangular Inequalities", Publ. Math. Debrecon, 8(1961), 169-186.

6. I. Burhan Turksen, "Interval Valued Fuzzy Sets Based on Normal Forms", Fuzzy Sets and Systems, 20(1986), 191-210.

7. I. Burhan Turksen, "Four Methods of Approximate Reasoning with Interval- Valued Fuzzy Sets", International Journal of Approximate Reasoning, 3, 2, 1989, 121-142

8. Sailesh K. Rao and Thomas Kailath, "Regular Iterative Algorithms and Their Implementation on Processor Arrays", Proc. IEEE, 76, 3, March 1988, 259-269

9. T. Yamakawa, "High-Speed Fuzzy-Controller Hardware System: The Mega FIPS Machine", Information Sciences, 45, 1988, 161-180

10. C.C. Lee, "Fuzzy Logic in Control Systems: Fuzzy-Logic Controller, Parts I and II", IEEE Trans. Systems, Man, and Cybernetics, 20, 2, 1990, 404-435
2

Acknowledgements

The authors wish to acknowledge the support of the Manufacturing Research Corporation of Ontario (MRCO) through research contracts in the areas of Management and Automation