UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

# *Programmable Finite Automata for VLSI*

*K. Culik II*
*H. Jürgensen*

*CS-82-24*

*September, 1982*

# PROGRAMMABLE FINITE AUTOMATA FOR VLSI†

*K. Culik II*

*H. Jürgensen*‡

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

## 1. The Aims

This paper presents several observations whose combination will lead to an outline of a quite powerful design method for VLSI-realizations of programmable finite automata. The automaton realization to be described will have the following properties:

(1) For any given fixed $k$ it can be programmed to perform the task of any non-deterministic automaton with at most $k$ states.

(2) There is a constant $c$ such that it runs in time $\frac{ck}{b}n + ck \log b$ on inputs of length $n > b$ and in time $ck \log b$ on those of length $n \leq b$. Logarithms are with respect to base 2. $b$ is a constant, influencing the size of the realization, which can be chosen such that $\frac{ck}{b} \ll 1$.

(3) It is made from uniform systolic components connected in a uniform manner. This will simplify the actual lay-out.

(4) It is modular, that is chips of this kind may be connected in order to increase the constant $b$ and thus decrease the time on long inputs. The constant $b$ can be chosen after the chips have been made.

† This work was supported by the Natural Sciences and Engineering Research Council of Canada under the grant A-7403.

‡ On a leave of absence from Institut für Theoretische Informatik, Technische Hochschule Darmstadt, Germany.

(5) It can be modified to accept prefixes of the input string whenever an accepting state is reached as an intermediate state.

(6) It contains $O(bk^2)$ essentially uniformly connected and identical processors.

Property (5) is particularly useful in string matching and retrieval applications, which are one of the most popular application areas of finite automata in practical problems [11]. It is also common with this type of problem that the data item to be searched for has a fairly short description, in terms of a finite automaton A, say, whereas the data file $f$ to be searched may be quite long. In many practical situations a bound on the size of A would be acceptable. However, fixing the structure of A, too, would certainly reduce the applicability of a hardware realization of A considerably. Our realization avoids this disadvantage by having A programmable; in addition the size bound on A is rather weak in most cases as it concerns the non-deterministic description of $A$. On the other hand by chosing the constant $b$ large enough we achieve a considerable speed-up of searching $f$ if $f$ is large. Finally, modularity of (3) will allow the user to increase the speed of a given system which uses these modules by adding same more modules and thus increasing $b$.

In the context of VLSI design the term "programmable" has also been used in a completely different sense. So in Foster and Kung [1] as well as in Floyd and Ullman [2] "programmability" means the existence of a programming language, that of regular expressions, say, and a uniform design method, a compiler, which given a regular expression produces the lay-out of the corresponding acceptor chip. This type of "programmability" has been referred to as uniform design above. In this paper, however, by programmability we mean the possibility to have the actual chip programmed at application rather than design time.

Designs of finite automata aimed at a VLSI realization have been considered in several papers before.

Foster and Kung [1] suggest a translation of regular expressions into VLSI. They use standardized components for comparison with single letters and for the operations of + and *. Concatenation is implemented by sequential connections. If $r$ is a regular expression and $A_r$ is its VLSI realization obtained from using this translation, then the size of $A_r$ is determined by the length of $r$, and $A_r$ takes $c_1 k_r n$ steps, approximately, for recognizing a word $f$ of length $n$. Here $c_1$ is the basic switching time and $k_r$ is a constant essentially equal to the "longest concatenation" in the expression $r$. Once manufactured as a chip, $A_r$ is programmable in a very restricted sense only; by preloading the letter comparison units appropriately it is programmed to accept any language which is definable by a

regular expression having the same structure as $r$ .

Its uniform design method as well as little area consumption - the area required is of the order $k_r \log k_r$ - are the merits of this solution. Its main disadvantages are its non-programmability as explained above, its highly complex though systematic wiring which depends on the "complexity" of $r$ , and the fact that essentially no use is being made of parallelism except for simulating the non-deterministic transitions of the automaton.

Floyd and Ullman [2] provide two solutions to the problem of designing integrated circuits for regular languages the first one being an implementation of a non-deterministic automaton on a programmable logical array (= PLA) and the second one using the hierarchical structure of the automaton obtained from the parse tree of the regular expression.

The solutions of [2], [1], and of this paper all attempt to arrive at a systematic design method which, essentially, could be automatized, that is, which given a description of a regular language, a regular expression, say, would eventually produce a VLSI-realizable design. In each case the circuit runs in linear time on long inputs; however, neither the implementation of [1] nor those of [2] can achieve a run time bounded by $c_0 n$ on inputs of length $n$ where $c_0$ is less than 1, as it always takes them at least $n$ time units to read the complete input. The hierarchical implementation of [2] is again of very little programmability as it implements the parse tree of the regular expression at hand. However, it seems that the PLA-version could be made fully programmable by adding appropriate gates at all intersections and also some new input connections which would allow one to set these gates according to the respective non-deterministic finite automaton.

Due to their low area requirements the implementations of [1] and [2] will be preferable to our solution when neither flexible programmability nor very high speed on very long inputs are the essentials, whereas for applications as string and pattern matching in large databases our approach might be more adequate.

A different type of hardware design method for string and pattern matching algorithms, whose description and review, however, is beyond the scope of the present paper, has been proposed by A. Mukhopadhyay [6].

## 2. Systolic Trees

In [3, 4] systolic trees are considered as acceptors. One of the main points of those papers is to explore the power and restrictions of certain simple VLSI topologies. Neither the realization of the processors involved nor the implications of

having a finite rather than an unbounded device are a central issue there.

A *systolic tree* [3]

$$K = \left(T\ ,\ \Sigma_P\ ,\ \nu\ ,\ \Sigma_0\ ,\ \Sigma'_0\ ,\ \Sigma_T\ ,\ \#\ ,\ \{g_A \mid A \in \Sigma_P\}, \{f_A \mid A \in \Sigma_P\}\right)$$

consists of

- an infinite tree $T$ whose nodes are labelled by the letters of the alphabet $\Sigma_P$ such that $T$ satisfies the growth, regularity, and arity conditions defined below,
- finite alphabets $\Sigma_0$, $\Sigma'_0 \subseteq \Sigma_0$, $\Sigma_T$ with $\Sigma_0$, $\Sigma_T$ non-empty, and $\#$ a special symbol in $\Sigma_T$,
- an arity function $\nu: \Sigma_P \to \mathbb{N} : A \mapsto \nu(A) \geq 1$,
- families of functions $g_A: \Sigma_T \to \Sigma_0$ and $f_A: \Sigma_0^{\nu(A)} \to \Sigma_0$.

$T$ is said to satisfy the *growth condition* if there is a constant $\alpha > 1$ such that the number of nodes at a distance $k$ from the root is no less than $\alpha^k$. It satisfies the *regularity condition* if it contains only finitely many non-isomorphic labelled subtrees. Finally, the *arity condition* holds on $T$ if for each node the number of sons is the same as the arity of its label.

$\Sigma_P$ is to be considered as the collection of *processor types*. $\Sigma_0$ is the *operating alphabet*, and a processor of type $A$, i.e., a node labelled $A$, would take the $\nu(A)$ outputs of its sons $x_1, \ldots, x_{\nu(A)} \in \Sigma_0$ and compute $f_A(x_1, \ldots, x_{\nu(A)}) \in \Sigma_0$ as its own output.

An input word $w$ over the *terminal alphabet* $\Sigma_T$ is fed into $K$ on an appropriate layer as follows: Let $k$ be the smallesst distance from the root such that there are at least $|w|$ nodes having this distance, where $|w|$ denotes the length of $w$. Let $A_1, A_2, \ldots, A_r$ be the nodes at the distance $k$, ordered from left to right in the natural way, and let $w = a_1, \ldots, a_{|w|}$, $a_i \in \Sigma_T$; then for $i = 1, \ldots, r$ the node $A_i$ will have

$$g_{A_i}(a_i) \quad \text{for} \quad i \leq |w| \quad \text{and} \quad g_{A_i}(\#)$$

as its output. Thus $g_A$ is the *input encoding* corresponding to $A$. The word is then processed towards the root as described above. $w$ will be *accepted* if - after $k$ steps - the root produces an output in the *accepting alphabet* $\Sigma'_0$. Let $L(K)$ denote the language accepted by $K$.

In [3] it is proved that, even though the isomorphism type of $T$ is quite important in determining the class of languages accepted by $K$, as far as regular languages are concerned it does not make an essential difference.

**Proposition 1.** [3] *Let $K$ be an arbitrary systolic tree with first component $T$. Then for every regular language $R$, there is a systolic tree $K_R$ with first component $T$ which accepts $R$.*

Thus for accepting regular languages we may consider complete binary trees only. Furthermore, as was shwon in [3], $\nu$ can always be made injective, i.e., in the case of complete binary trees $|\Sigma_P| = 1$. Thus a *binary systolic tree* is a 7-tuple

$$K = (T, \Sigma_0, \Sigma'_0, \Sigma_T, \#, g, f)$$

with $T$ a complete infinite binary tree such that

$$(T, \{A\}, \nu: A \mapsto 2, \Sigma_0, \Sigma'_0, \Sigma_T, \#, \{g_A = g\}, \{f_A = f\})$$

is a systolic tree.

**Proposition 2.** [3] *Each regular language is accepted by a binary systolic tree.*

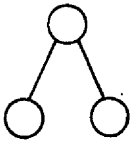
The proof of this statement uses an idea which will serve as one basis of our construction. Therefore we give a brief outline of it: Let $R \subseteq \Sigma^*$ be a regular language and let $Syn(R)$ be the syntactic monoid of $R$. Let $\Sigma_0 = Syn(R)$, $\Sigma_T = \Sigma \cup \{\#\}$, $g(a) = [a]$ for $a \in \Sigma$, where $[a]$ is the syntactic class of $a$, $g(\#) = [\varepsilon]$ with $\varepsilon$ the empty word, $\Sigma'_0$ the set of syntactic classes of words in $R$, and $f([u],[v]) = [uv]$.

In this construction each node simulates the complete combinatorial structure of $R$, and $\#$ plays the role of an empty input. It is, therefore, clear that this binary systolic tree is even *super-stable* [4], i.e., essentially it does not matter whether it receives its input on the lowest possible layer nor whether the input is arbitrarily interrupted by sequences of $\#$ symbols.

A binary systolic tree recognizes its input $w$ in $k\lceil \log |w| \rceil$ steps where $k$ is proportionnal to the number of basic steps needed to compute the functions $f$ and $g$.

The logarithmic time is attractive, but, of course, not achievable by finite realizations. With respect to realizations we shall restrict to finite tree structures rather than unbounded ones and also explore the structure of the nodes to a certain extent.

The construction of the proof of Proposition 2 may be modified in order to reduce the complexity of the nodes in terms of the size of that part of $\Sigma_0$ which is actually needed for recognition: Consider the binary tree



and suppose that in its root it computes the product of its inputs $m_1, m_2$, elements of the monoid $M$, and that $m_1m_2$ is accepted if $m_1m_2 \in G \subseteq M$. Evidently it is relevant for the root to know what $m_1$ really is only if

$$m_1 \in GM^{[-1]} = \{m \mid mM \cap G \neq \phi\}$$

and similarly for $m_2$ only if

$$m_2 \in M^{[-1]}G = \{m \mid mM \cap G \neq \phi\} .$$

Hence let

$$M_L = GM^{[-1]} \cup O_L , \quad M_R = M^{[-1]}G \cup O_R$$

where $O_L$, $O_R$ are new elements, such that, in fact, the left and right sons of the root have $M_L$ and $M_R$ as their output sets, respectively. Then the root would evaluate the mapping

$$M_L \times M_R \to G \cup O: \begin{cases} (m_1, m_2) \mapsto m_1m_2 & \text{if } m_1 \neq O_L,\ m_2 \neq O_R \\ & \text{and } m_1m_2 \in G , \\ (m_1, m_2) \mapsto O & \text{if } m_1 = O_L \text{ or } m_2 = O_R \\ & \text{or } m_1m_2 \notin G. \end{cases}$$

Applying this construction recursively to a binary systolic tree for a regular language $R$ yields a tree in which each of the nodes uses as little of $\Sigma_0$ as possible: For the root let $G = g(R)$; then if $G$ is given for some node in the tree let $G_L = GM^{[-1]}$, $G_R = M^{[-1]}G$ the corresponding sets for its left and right sons, respectively, with $M = Syn(R)$ and $g$ the syntactic homomorphism.

**Example.** Consider $R = a^3b^*c\{a, b\}$. Then $M = SynR$ consists of the 8 classes $[\varepsilon]$, $[a]$, $[a^2]$, $[a^3]$, $[b]$, $[ab]$, $[a^2b]$, $[ba]$, and the image of $R$ is $G = \{[a^3]\}$. Then

$$GM^{[-1]} = \{[\varepsilon],[a],[a^2],[a^3]\} \ .$$
$$M^{[-1]}G = M \setminus \{[ba]\} \ .$$

Continuing this we obtain

$$(GM^{[-1]})M^{[-1]} = GM^{[-1]}$$

and

$$M^{[-1]}(GM^{[-1]}) = M \setminus \{[ba]\} \ .$$

and of course

$$(M^{[-1]}G)M^{[-1]} = M^{[-1]}(M^{[-1]}G) = M \setminus \{[ba]\} \ .$$

As output sets of the nodes in the tree we obtain

$G \cup O = \{[a^3], O\}$ for the root,
$GM^{[-1]} \cup O = \{[\varepsilon],[a],[a^2],[a^3], O\}$ for all left-most nodes, and
$M$ for all other nodes.

One easily proves that, in general, if $M = Syn R$ for a language $R \subseteq X^*$ and $g(R) = G$ for $g$ the syntactic homomorphism of $X^*$ onto $M$, then

$$GM^{[-1]} = (GM^{[-1]})M^{[-1]} \ , \quad M^{[-1]}G = M^{[-1]}(M^{[-1]}G) \ ,$$

and

$$M^{[-1]}(GM^{[-1]}) = (M^{[-1]}G)M^{[-1]} = \overline{M}$$

where

$$\overline{M} = \begin{cases} M & \text{if } M \text{ has no zero} \ . \\ M\ O & \text{if } M \text{ has a zero} \ . \end{cases}$$

## 3. Finite Binary Systolic Trees with Feed-Back

Clearly, with the input conventions of systolic trees, a finite tree can only accept a finite languages. In order to arrive at a more interesting and more realistic design we shall therefore modify those rules.

A *finite binary systolic tree* is a 7-tuple

$$K = (T, \Sigma_0, \Sigma'_0, \Sigma_T, \#, g, f)$$

where $\Sigma_0, \Sigma'_0, \Sigma_T, \#, g, f$ as before; $T$ is a finite tree whose nodes are at most binary; binary nodes are labelled $f$, unary

nodes are labelled *id* for the identity mapping; leaves don't need a label as they serve as (the only) input nodes.

If $K$ is a finite binary systolic tree let $b$ be the number of leaves of $K$. Let $\Sigma = \Sigma_T \{\#\}$. A word $w \in \Sigma^*$ of length $n = |w|$ is processed by $K$ as follows: Decompose $w$ into $r = \lceil \frac{n}{b} \rceil$ blocks $b_1 b_2 \cdots b_r = w$ with $|b_1| = |b_2| = \cdots = |b_{r-1}| = b$, $0 < |b_r| \leq b$. Then at successive epochs enter $g(b_1), g(b_2), \ldots, g(b_r)$ at the leaves of $T$ with $b_r$ possibly expanded to length $b$ by an appropriate number of # symbols.

Thus, the computations on $b_1, \ldots, b_r$ travel through the tree until they reach the root. In order to combine their results we add a new "root" node with a feed-back loop as shown in Figure 1.

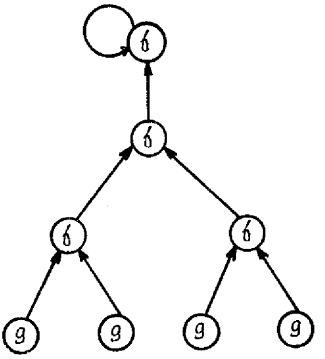



Figure 1

Let $K^1$ denote this *finite binary systolic tree with feed-back* obtained from $K$. By unrolling the computation of $K^1$ [8], we obtain the following infinite tree (Figure 2) which can be considered as determining the structure of an equivalent (infinite) systolic tree.

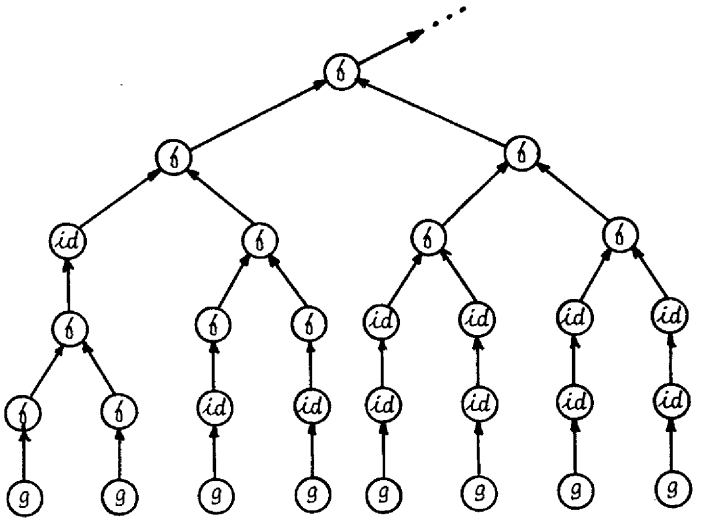



Figure 2

By superstability (or directly, of course) one obtains:

**Proposition 3.** *For each regular language* $R$ *and each* $b \in N$ *there is a finite binary systolic tree with feed-back* $K^t$ *having the following properties:*

(1) $K^t$ *accepts* $R$ .

(2) *K's tree has* $b$ *leaves.*

(3) $K^t$ *needs approximately* $\lceil \frac{n}{b} \rceil + \lceil \log b \rceil$ *steps for recognizing words of length* $n$ .

(4) $K^t$ *consists of* $b$ *nodes realizing* $f$ *and* $2^{\lceil \log b \rceil} - b$ *nodes realizing id* .

From the above remarks the statement is clear for $b$ a power of 2. Otherwise let $h = \lceil \log b \rceil$ and consider a binary tree $T'$ of height $h$ with $b$ leaves. Extend all branches of $T'$ whose height is less than $h$ by an appropriate number of unary nodes. The resulting tree $T$ would be used in $K^t$ . For example, for $b = 5$

one would have Figure 3.

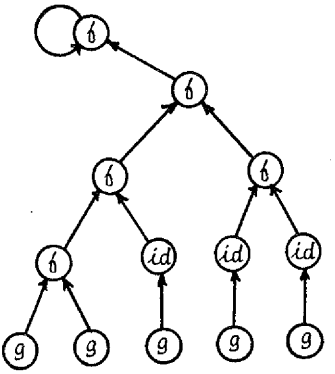



Figure 3

As an immediate consequence of these remarks we also obtain a composition operation for finite binary systolic trees: Let

$$K_1 = (T_1, \Sigma_0, \Sigma'_0, \Sigma_T, \#, g, f)$$

and

$$K_2 = (T_2, \Sigma_0, \Sigma'_0, \Sigma_T, \#, g, f)$$

be finite binary systolic trees and let $\lambda$ be a leaf of $T_1$. Then

$$K_1 \overset{\circ}{\lambda} K_2$$

denotes the finite binary systolic tree

$$K = (T, \Sigma_0, \Sigma'_0, \Sigma_T, \#, g, f)$$

whose underlying tree $T$ is obtained from $T_1$ and $T_2$ by replacing $\lambda$ by $T_2$ and then again making all branches of the same height by adding appropriately many unary nodes. For an example see Figure 4. Obviously the construction can be

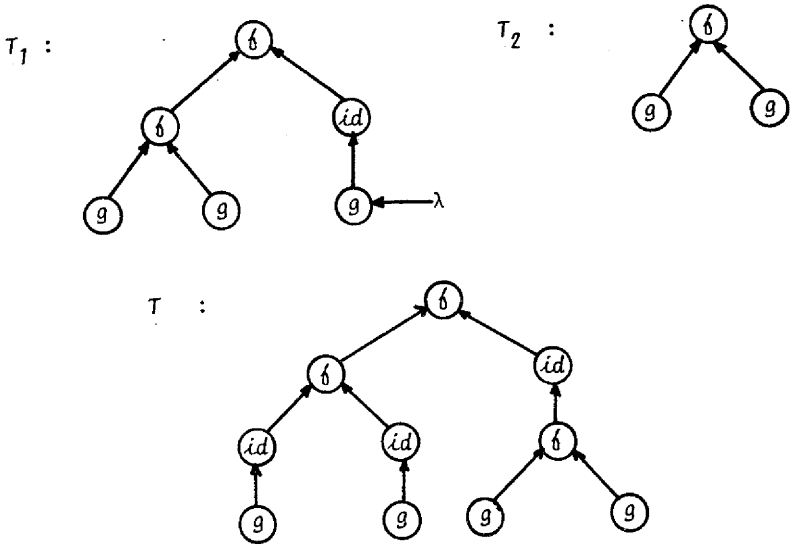



**Figure 4**

modified in such a way that also unary nodes can be selected as replacement destinations $\lambda$. This simple observation is the key to modularity, efficiency, and combinability as outlined in §1(4) as it helps avoiding to introduce redundant delays by unary nodes.

**Proposition 4.** *Let* $K_i = (T_i, \Sigma_0, \Sigma'_0, \Sigma_T, \#, g, f)$ *be finite binary systolic trees for* $i = 1,2$, *and let* $\lambda$ *be a leaf of* $T_1$ *or a unary node of* $T_1$ *such that the subtree of* $T_1$ *with root* $\lambda$ *is a single branch. If* $K_1^1$ *accepts the regular language* $R$, *then* $(K_1 \overset{\circ}{\lambda} K_2)^1$ *also accepts* $R$.

By Proposition 4 an acceptor for a regular language $R$ can be obtained as a composition of several identical modules (chips, e.g.), each $a$ finite binary systolic tree with $b$ inputs, where, for convenience, $b$ should be a power of 2. Such an acceptor would even be expansible by simply adding a few more of these

components to it. Observe that these additions would increase the number of leaves for input and thus the speed on long inputs; however, it would slightly decrease the speed on short inputs, unless the old input nodes remain accessible also.

## 4. The Nodes

Up to now we were mainly concerned with global properties of finite binary systolic trees with feed-back. We now turn to considering the realization of the nodes and some aspects of the timing connected to this.

Let $R \subset \Sigma^*$ be a regular language, let $M = Syn(R)$ be its syntactic monoid, and let $G \subseteq M$ be the image of $R$ under the natural homomorphism of $\Sigma^*$ onto $M$. For an efficient realization of the nodes and a simplified design we need a representation $\varphi$ of $M$ in a concrete (= itself represented) monoid $B$, that is, a homomorphism $\varphi$ of $M$ into $B$ and an evaluation function $val: B \to \mathbf{B} = \{0, 1\}$ such that

$$val(\varphi(m)) = 1 \leftrightarrow m \in G .$$

Here $\mathbf{B}$ denotes the Boolean semiring

There are several obvious candidates for $B$ :

- $M$ itself given by a Cayley table,
- the transition monoid of a deterministic or non-deterministic acceptor of $R$
- the monoid $\mathbf{B}_n$ of $n \times n$-matrices over $\mathbf{B}$ for an appropriate $n$ .

The first possibility is ruled out by space and time considerations - keeping in mind that the network should be programmable to accept various regular languages. In the second case we would need a representation of the mapping of the state set into itself; this could be achieved by tables for graphs thus introducing space and time problems agin, or by Boolean matrices which in fact gives the third possibility. In the sequel we therefore consider that one only.

As is well-known [9], a language $R$ over $\Sigma$ is regular if and only if there is $n \in \mathbf{N}$ , a representation $\varphi$ of $\Sigma^*$ in $B_n$ , a row vector $\pi \in \mathbf{B}^{1 \times n}$ , and a column vector $\eta \in \mathbf{B}^{n \times 1}$ such that for all $w \in \Sigma^*$

$$w \in R \quad \text{if and only if} \quad \pi\varphi(w)\eta = 1 .$$

In this statement we may consider $n$ as the number of states of a non-deterministic acceptor of $R$, $\{i \mid \pi_i = 1\}$ as the non-

deterministic initial state, $\{i \mid \eta_i = 1\}$ as the set of final states, and for $\varphi(a)$ for $a \in \Sigma$ as the transition matrix corresponding to $a$.

Using this representation multiplication in $Syn(R)$ reduces to matrix multiplication in $B_n$, and the evaluation val is obtained by two vector multiplications. For these the multiplication networks of [5] could be used. For $n = 3$, a typical node is given in Figure 5 where each of the processors computes $xy + z$ from its inputs $x, y, z$ and where the $c_{ij}$'s are initially 0.

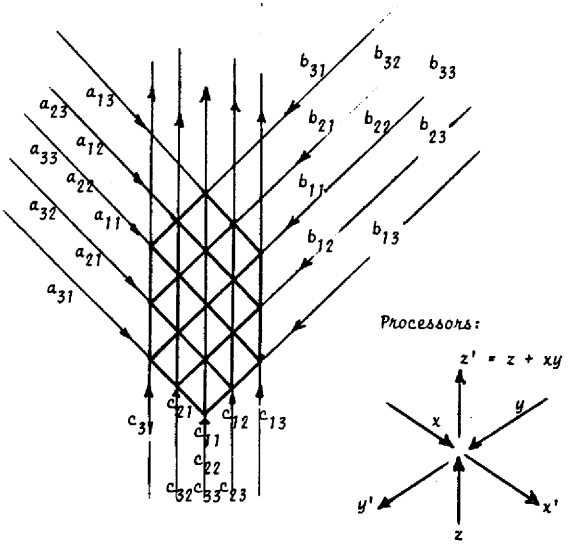



Figure 5

For a realization of a finite binary systolic tree with feed-back $K^{1,n}$ we now fix $n$, the dimension of the matrices. However, we don't fix the representation $\varphi$ nor the evaluation vectors $\pi, \eta$. This is feasible as, given an appropriate encoding of $\Sigma$ into $B_n$, none of the interior nodes need to have any additional external information about the representation. A typical application of $K^{1,n}$ would run as follows:

(1) $R$ is specified by a non-deterministic acceptor $A$ with non-deterministic initial state accepting $R$ and having no more than $n$ states (for reduction and minization of non-deterministic automata see [7]). Of course one may consider other specifications of $R$ from which $A$ would be obtained as usual. From $A$ determine $\pi, \eta$, and the transition matrices for the symbols. The latter will be looked up by the input encoding $g$.

(2) Break the input word $w$ into blocks $b_1, \ldots, b_r$ as outlined before and feed them into $K^{t,n}$. For doing so, for block $b_i$ say, the matrices corresponding to the symbols in $b_i$ will be pipelined into the corresponding leaves with the sequence as shown in Figure 5.

(3) The output of the feed-back node, $\varphi(b_1 \cdots b_i)$ for $i = 1, \ldots, r$, besides being an input to this node is also sent to an additional *evaluation node*, which computes $\pi\varphi(b_1 \cdots b_i)\eta$.

This proves:

**Proposition 5.** *Let $K^{t,n}$ be the realization of a finite complete binary systolic tree with feed-back and evaluation. $K^{t,n}$ has the following properties:*

(i) *For any choice of matrices in $B_n$ encoding the input and for any $\pi, \eta$ the language accepted by $K^{t,n}$ is regular.*

(ii) *$K^{t,n}$ accepts any regular language which has a non-deterministic acceptor with no more than $n$ states and with possibly more than one initial state.*

(iii) *For an input word $w$ the time needed is approximately*

$$c \cdot (5n-1)\left\lceil \frac{b}{|w|} \right\rceil + \log b$$

*where $b$ is the number of leaves of $K^{t,n}$.*

(iv) *$K^{t,n}$ consists of $(3(n-1)n+1) \cdot b$ processors for the matrix products and $\frac{3}{2}n^2 + \frac{n}{2}$ processors for the evaluation node each computing $xy + z$ from its Boolean inputs $x, y, z$.*

An example of $K^{t,n}$ for $n = 3$ is shown in Figure 6, for $b = 2$, which is unrealistically small, of course.

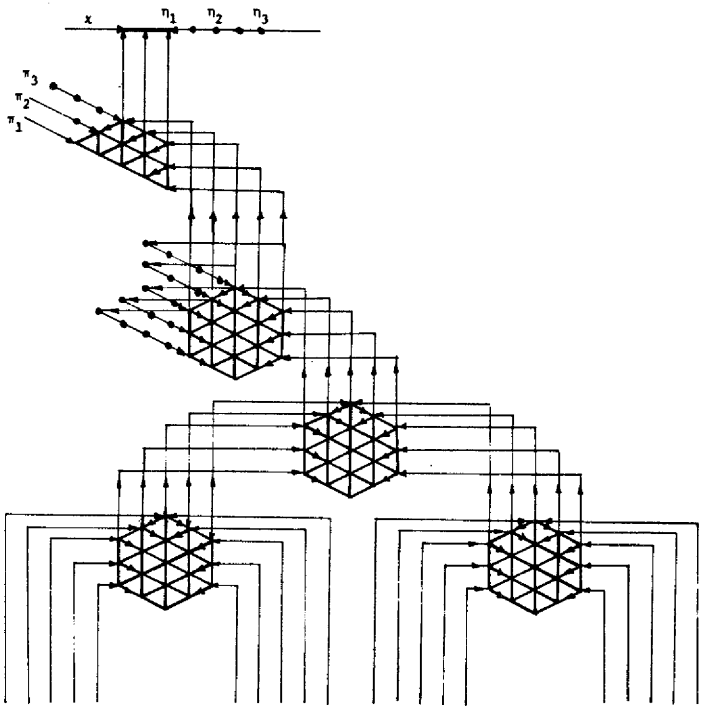



Figure 6

The matrix product circuit of [5] which was used here turns out to have two undesirable properties when it is cascaded like this:

(1) The output sequence has to be twisted once when it is to be used as input to the next stage.

(2) Inputs enter where outputs leave so that comparatively long connections are needed.

The second problem could be overcome by rearranging the inputs as shown in Figure 7.

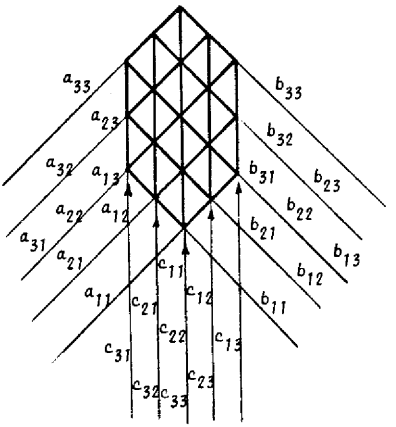



Figure 7

With this scheme the multiplication of $n \times n$-matrices becomes even faster taking $3n-1$ steps; but, connecting the output to another circuit of this kind is even more complicated.

However, a solution to both problems can be obtained using the technique of folding as outlined in [8]. In fact, by folding the circuit of Figure 5 twice along the dotted lines shown in Figure 8, one arrives at a circuit which allows for short connections to the next stage and avoids having to twist the outputs.

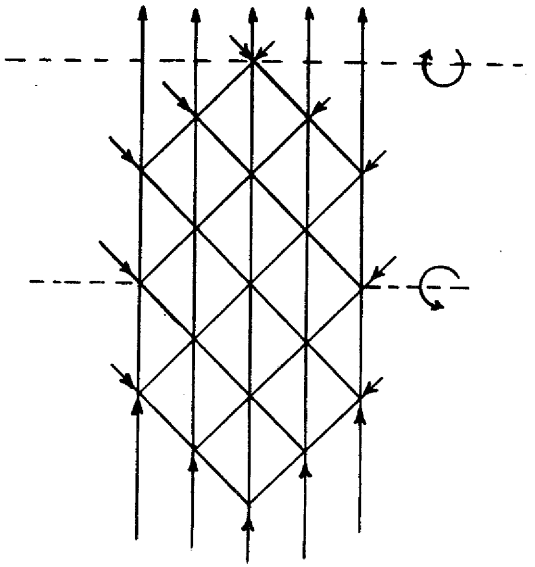


Figure 8

**References**

[1] M.J. Foster, and H.T. Kung: Recognize regular languages with programmable building blocks. 75-84 in J.P. Gray (ed.), *VLSI 81*. Academic Press, New York, 1981.

[2] R.W. Floyd, and J.D. Ullman: The compilation of regular expressions into integrated circuits. *FOCS '80*, 260-269.

[3] K. Culik II, A. Salomaa, and D. Wood: *VLSI systolic trees as acceptors*. Research Report CS-81-32, Department of Computer Science, University of Waterloo, 1981.

[4] K. Culik II, J. Gruska, and A. Salomaa: *Systolic automata for VLSI on balanced trees*. Research Report CS-82-01, Department of Computer Science, University of Waterloo, 1982.

[5] C. Mead, and L. Conway: *Introduction to VLSI Systems*. Addison-Wesley Publ. Co., Reading, Mass. 1980.

[6] A. Mukhopadhyay: Hardware algorithms for nonnumeric computation. *IEEE Trans. on Computers*, C-28 (1979), 384-394.

[7] P.H. Starke: *Abstrakte Automaten*. VEB Deutscher Verlag der Wissenschaften, Berlin, 1969.

[8] K. Culik II, and J. Pachl: Folding and unrolling systolic arrays. *ACM Symposium on Principles of Distributed Computing*, Ottawa, 1982.

[9] A. Salomaa and M. Soittola: *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, Berlin, 1978.

[10] M. Steinby: *Systolic Trees and Systolic Language Recognition by Tree Automata*. Preprint, Turku, 1982.

[11] A.V. Aho: Pattern matching in strings. 325-348 in R.V. Book (ed.), *Formal Language Theory*. Academic Press, New York, 1980.