Philips Research Laboratory
Av. Van Becelaere 2 - Box 8
B- 1170 Brussels, Belgium

Report R444

Substitution Networks

C. RONSE

October 1980

Abstract

We study general properties of switching networks realizing substitutions, i.e. bijections between their input and output contents. We give several designs for networks realizing all substitutions on a given number of bits, either by discrete components, or by expansion methods using smaller substitution networks and canonical universal logic modules. We compare their respective costs and delays in terms of discrete components. We generalize some known properties on variable dependency.

Finally we make a link between the expansion of substitution networks, the DES encryption algorithm and nonautonomous feedback shift registers.

SUBSTITUTION NETWORKS

I. Introduction

A substitution network is a switching network which can encypher a text of bounded length written in a given alphabet into another text of the same length written in the same alphabet.

We represent it in Figure 1.

Suppose that the text has length n and that the alphabet is a set K. If K has size m, then we can identify K with the set $\{0,1,\dots,m-1\}$. We need to take a switching network S with n data inputs $I_0,\dots,I_{n-1}$, n data outputs $O_0,\dots,O_{n-1}$ and k control inputs $C_0,\dots,C_{k-1}$. Data inputs and outputs take their values in the alphabet K, while control variables take their values in an alphabet D. We will often take D=K.

The contents $[x_{n-1},\dots,x_0]$ of the data inputs forms the plaintext, the ones $[y_{n-1},\dots,y_0]$ of the outputs form the cyphertext, and the ones $[c_{k-1},\dots,c_0]$ of the control variables form the key.

Given a key $[c_{k-1},\dots,c_0]$, S realizes the corresponding encypherment, that is a bijection :

$$f_{[c_{k-1},\dots,c_0]} = f : K^n \to K^n \qquad (1)$$

from its input contents to its output contents.

We need to take a bijection, because the encypherment must be one to one (in order to allow the decypherment).

The bijection f is called a substitution. The number of possible substitutions that S can realize is at most :

$$\min\{(|K|^n)!,\ |D|^k\}, \qquad (2)$$

because $(|K|^n)!$ is the number of substitutions $K^n \to K^n$ and $|D|^k$ is the number of different keys $[c_{k-1},\dots,c_0]$.

Let us make a few definitions : we will say that n is the <u>size</u> of S. The number of substitutions that S can realize will be called the <u>multiplicity</u> of S. If the multiplicity of S is $(|K|^n)!$ (in other words if S generates all substitutions), then we will say that S is <u>total</u> or that S has <u>maximal multiplicity</u>.

To S corresponds a second substitution network S^1 with the same text and key lengths as S, but which for a given key $c_{k-1},\dots,c_0$, performs the inverse substitution $(f_{[c_{k-1},\dots,c_0]})^{-1}$. S^1 is used to <u>decypher</u> the texts encyphered by S, and is called the <u>inverse substitution network</u> of S.

The main problem concerning this topic is the design of substitution networks which :

1) can perform certain prescribed substitutions.
2) can be built in practice with the present technology (this is specially the case when n is large).
3) have relatively low cost and delay.
4) have a relatively easy control algorithm.

The literature on this subject is relatively scarce (see the references [2,3,4,5,6] and related papers).

II. Some designs of substitution networks in the binary case

We suppose that $D=K=\{0,1\}$, in other words that we work in binary logic. We want to build a substitution network of size n.

A. The classical direct design.

The three basic elements of this design are the address decoder, the address encoder and the (partial) permutation network.

This design is summarily described in [2].

Given a number $k \in \{0,\dots,2^n-1\}$, let $[k_{n-1},\dots,k_0]$ be its binary representation (with $k_i \in \{0,1\}$ for each i).

An n to 2^n address decoder has n inputs $x_0,\dots,x_{n-1}$, 2^n outputs $y_0,\dots,y_{2^n-1}$, and realizes the following input-output behavior :

For every $k \in \{0,1,\dots,2^n-1\}$,

$$y_k = \prod_{i=0}^{n-1} x_i^{(k_i)}, \tag{3}$$

where we define :

$$\begin{aligned} a^{(b)} &= a \oplus b \oplus 1 = a \text{ if } b=1. \\ &= \bar{a} \text{ if } b=0. \end{aligned} \tag{4}$$

It follows that if $[x_{n-1},\dots,x_0] = x$, then :

$$\begin{aligned} & y_x = 1 \\ \text{and} \quad & y_k = 0 \text{ for } k \neq x. \end{aligned} \tag{5}$$

A 3 to 8 address decoder is shown in Figure 2.

A 2^n to n address <u>encoder</u> is a switching network with 2^n inputs $v_0,\dots,v_{2^n-1}$ and n outputs $z_0,\dots,z_{n-1}$, which can realize the inverse of the input-output behavior of the n to 2^n address decoder. Thus (cfr. (5)), if :

$$v_x = 1$$

and

$$v_k = 0 \text{ for } k \neq x,$$

then

$$[\, z_{n-1},\dots,z_0 \,] = x. \tag{6}$$

Of course, the behavior of the address encoder is uncompletely specified. We know the values of the outputs only for the input vectors of weight 1.

One possible choice for complete functions satisfying (6) is :

$$z_i = \bigvee_{k_i=1} v_k, \tag{7}$$

where $i=0,\dots,n-1$, $k \in \{0,\dots,2^n-1\}$ and $[\, k_{n-1},\dots,k_0 \,] = k$.

Indeed, if we take $v_x = 1$ and $v_k = 0$ for $k \neq x$, then (7) becomes

$$\begin{aligned} z_i &= v_x \quad \text{if} \quad x_i=1, \\ &= 0 \quad \text{otherwise,} \end{aligned} \tag{8}$$

in other words

$$z_i = x_i, \tag{9}$$

and so (6) holds.

An 8 to 3 address encoder is shown in Figure 3, with the values of the outputs z_i obtained by (7).

Note that in (7) the variable v_0 is idle.

A partial permutation network on m bits is a switching network with m data inputs, m outputs (and a certain number of control inputs), which can perform some permutations of the signals (see [11]). A permutation network is a partial permutation network which can perform all such permutations [11].

The design is the following. Take an n to 2^n address decoder $\mathcal{D}$, a partial permutation network $\mathcal{P}$ on 2^n bits and a 2^n to n address encoder $\mathcal{E}$ and connect them in series as is shown in Figure 4 for n=3. Then we get a substitution network $\mathcal{S}$ having the same multiplicity as $\mathcal{P}$.

Indeed, to any substitution $f : K^n \to K^n : (x_{n-1},\dots,x_0) \to (x^1_{n-1},\dots,x^1_0)$ corresponds the permutation $\pi(f)$ of $\{0,1,\dots,2^n-1\}$ which maps $x = [x_{n-1},\dots,x_0]$ on $x^1 = [x^1_{n-1},\dots,x^1_0]$.

If $\mathcal{P}$ realizes $\pi(f)$, then the network $\mathcal{S}$ realizes f. In particular, if $\mathcal{P}$ is a permutation network, then $\mathcal{S}$ is total.

This type of design can be generalized to the nonbinary case, provided that one gives a suitable definition of address decoders and encoders.

B. Universal Logical Modules.

It is possible to get all substitutions of size n by taking a switching circuit which can realize all functions $K^n \to K^n$. The increase in cost is not substantial because the logarithms of the number of substitutions of size n and of the numbers of functions $K^n \to K^n$ are both of the form $O(n2^n)$.

A switching network with n data inputs, k control inputs and m outputs, which, under a proper setting of the control inputs, can realize any function $f : K^n \to K^m$ from its data inputs to its outputs will be called an n to m canonical universal logical module (or CULM) (see [12]).

As there are $(2^m)^{2^n}$ functions $K^n \to K^m$, we may take :

$$k = m2^n \tag{10}$$

A design of an n to m CULM using m multiplexers with the same n control variables is shown in Figure 5. For any $k \in \{0,\ldots,2^n-1\}$ and $j \in \{0,\ldots,m-1\}$, $c_k^{(j)}$ is the value taken by y_j when $[x_{n-1},\ldots,x_0] = k$.

As the multiplexers use the same control variables $x_0,\ldots,x_{n-1}$, the minterms that they use can be generated in the same address decoder. One gets thus the design shown in Figure 6 for n=2 and m=3.

Thus design is easier than the classical direct design, and has a simpler control algorithm. It can be generalized to the nonbinary case, provided that one gives a suitable generalization of the gates it uses.

III. The expansion of substitution networks

When n is large, the design of substitution networks of size n by the preceding methods may become very intricate and cannot be built on a single chip. It is therefore necessary to build substitution networks with the use of substitution networks of smaller size and maybe other components.

We will first give two methods corresponding to the two designs of the preceding section. Then we will indicate some possible directions for the study of this problem.

A. The expansion of CULM's

As an n to n CULM contains a total substitution network of size n, our problem can be solved by studying the building of "large" CULM's from "smaller" ones.

We will not restrict ourselves to the binary case. A CULM can be defined for every value of $|K|$. But then (10) becomes :

$$k = m.|K|^n \tag{11}$$

We will write $|K| = \lambda$.

An n to (m+t) CULM on K can be built with an n to m and an n to t CULM's, as is shown in Figure 7.

In the binary case, this design can be made at cheaper cost. If we consider the design of Figure 6, we see that an n to m CULM can be built by connecting the outputs of an n to 2^n address decoder to m copies of a circuit called an <u>enabler</u> on 2^n bits, illustrated in Figure 8 for n = 2. Then the two CULM's of Figure 7 can use the same address decoder and so we get the design of Figure 9.

Now let us consider the building of an (n+t) to m CULM from n to u and t to v CULM's (with u, v divisible by m).

Let $A=K^m$, $B=K^t$ and $C=K^n$. Then a function $f : B \times C \to A$ can be identified with a function $g : C \to A^B$, where A^B is the set of functions $B \to A$, and vice versa. The identification is made in the following way :

For any $f : B \times C \to A$ and $g : C \to A^B$, f can be identified with g if for any $c \in C$ and $b \in B$ we have :

$$f(b,c) = (g(c))(b). \tag{12}$$

(Here g(c) is a function $B \to A$ corresponding to c).

As $|A^B| = \lambda^{m\lambda^t}$, we can thus build an (n+t) to m CULM by connecting the outputs of an n to $m\lambda^t$ CULM to the control inputs of a t to m CULM.

This is shown in Figure 10. For a u to v CULM, we write $c_\gamma^{(\delta)}$ $(0 \leqslant \delta < v,\ 0 \leqslant \gamma < \lambda^u-1)$ for the control variables, where $c_\alpha^{(\beta)}$ is the value of the output labelled β when the input vector corresponds to α. (cfr the design of Figures 5 and 6 in the binary case). For any $\beta \in \{0,\dots,m\lambda^t-1\}$, we will write $\beta = [\beta_1,\beta_0]$ with $\beta_1 \in \{0,\dots,m-1\}$ and $\beta_0 \in \{0,\dots,\lambda^t-1\}$.

Now in Figure 10, every y_β is connected to the control input $c_{\beta_0}^{(\beta_1)}$ of the t to m CULM. We choose the following values for the control inputs of the n to $m\lambda^t$ CULM :

If $f(x_{n+t-1},\dots,x_0) = (\bar{z}_{m-1},\dots,\bar{z}_0)$, then set $c_\alpha^{(\beta)} = \bar{z}_{\beta_1}$

for $\beta_0 = [x_{n+t-1},\dots,x_n]$

and $\alpha = [x_{n-1},\dots,x_0]$ (13)

Indeed, if we make such a choice for the $c_{\alpha}^{(\beta)}$'s, then we get :

$$y_{\beta} = \bar{z}_{\beta_1} \quad \text{for} \quad \alpha = [\,x_{n-1},\ldots,x_0\,] \tag{14}$$

and so we have in the t to m CULM :

$$c_{\beta_0}^{(\beta_1)} = \bar{z}_{\beta_1}$$

$$\text{for} \quad \alpha = [\,x_{n-1},\ldots,x_0\,]$$

$$\text{and} \quad \beta_0 = [\,x_{n+t-1},\ldots,x_0\,] \tag{15}$$

But then we obtain :

$$z_{\beta_1} = \bar{z}_{\beta_1}$$

for $\alpha = [\,x_{n-1},\ldots,x_0\,]$

and $\beta_0 = [\,x_{n+t-1},\ldots,x_n\,]$,

and so the network realizes f.

Notes

1) This network can also be obtained if we expand the components of CULM's in the binary case, i.e. the address decoders and the enablers. However, the description is more intricate with that method.

2) If we use the design of Figure 5 in the binary case,then our expansion is equivalent to the tree expansion of multiplexers.

B. The three stage expansion of total substitution networks

In section II.A we designed a total substitution network with the use of a permutation network. The idea was to consider the 2^n different input vectors as points and to realize the substitution as a permutation of those points.

So we can hope to generalize the techniques for the expansion of permutation networks to substitution networks.

We will suppose that $|D| = |K| = k$ and even that $D = K = \{0,\dots,k-1\}$ (where k is an integer > 1).

We want to build a total substitution network of size n with substitution networks of smaller sizes and possibly other components.

This corresponds to the problem of building of permutation networks on k^n bits with the use of permutation networks on k^t bits, where $0 \leqslant t < n$.

It is clear (see [11]) that the permutation network design corresponding to that problem is the Clos Network which we describe below :

Let a,b be two integers such that $a,b > 1$. Let A and B be two permutation networks on a and b bits respectively. Then the Clos network $B \times\times A$ is illustrated in Figure 15 for a=3 and b=2. It consists of one stage of a copies of B, then one stage of b copies of A, and finally one stage of a copies of B, and the stages are connected by perfect shuffles.

The Clos network is a permutation network [11]. Let us explain it in terms of permutations :

Let $Z_a = \{0,\dots,a-1\}$ and $Z_b = \{0,\dots,b-1\}$. Then consider the permutations of $Z_a \times Z_b$. The first stage of the Clos network generates all permutations of the following type :

$$\pi : Z_a \times Z_b \to Z_a \times Z_b : (x,y) \to (x,y)\pi = (x,y(\pi_x)), \qquad (16)$$

where each π_x $(x \in Z_a)$ is a permutation of Z_b.

The second stage generates all permutations of the following type :

$$\tau : Z_a \times Z_b \to Z_a \times Z_b : (x,y) \to (x,y)\pi = (x(\pi_y),y), \tag{17}$$

where each π_y $(y \in Z_b)$ is a permutation of Z_a.

Finally, the third stage generates all permutations of the type (16).

Now the fact that the Clos network is a permutation network means that every permutation of $Z_a \times Z_b$ can be decomposed as the product of three permutations, the first one of type (16), the second one of type (17) and the third one of type (16).

Now, if we take $a=k^t$, $b=k^n$, $Z_a=K^t$ and $Z_b = K^n$ (with $n,t > 1$), then let us see what type of substitution corresponds to a permutation of type (16). It is a substitution on the first n bits of a vector, depending on the last t bits. It can be generated by the network illustrated in Figure 12 :

Here S is a total substitution network of size n and C a t to u CULM, where u is the number of control inputs of S. For every choice of $x_n,\ldots,x_{n+t-1}$, C determines the substitution on $x_0,\ldots,x_{n-1}$.

Now the three stage realization of a total substitution network corresponding to the Clos network is illustrated in Figure 13. Here S_n and S_t are total substitution networks of sizes n and t respectively, and C_t and C_n are respectively, a t to u and a n to v CULM's, where u and v are the numbers of control inputs of S_n and S_t respectively.

The control algorithm of this design is similar to the one used for the Clos Network [1,7,8,9,10,13,14]. When k=2 and n=1, the "looping" algorithm defined in [10] can be used. We will explain it on the following example :

Let us take t=2 (we have k=2 and n=1). Then Figure 13 reduces to Figure 14. Consider the substitution f defined by :

$$y_0 = x_0 \oplus x_1x_2.$$
$$y_1 = x_0 \oplus x_1 \oplus x_2 \oplus x_0x_2 \oplus 1.$$
$$y_2 = x_0 \oplus x_1 \oplus x_0x_2. \qquad (18)$$

It has the following thruth table :

x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	1	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	1	1	1
1	1	1	1	1	0

(19)

Let us start our procedure. We will put the input and output contents in pairs : every (a,b,c) will be paired with $(a,b,\bar{c})$. This corresponds to the pairs of inputs and outputs of the 2-cells used in the Clos network when b=2.

Let us take first $(x_2,x_1,x_0)=(0,0,0)$. It is mapped by f on $(y_2, y_1,y_0)=(0,1,0)$. We will choose arbitrarily the value of the control input c in this case. The easiest is to choose c=0 (and so $z_0=x_0$)(*). We have thus this first fact :

For $(x_2,x_1)=(0,0)$,

$$c=0. \qquad (20)$$

(*) See Figure 14 for the definition of z_0, c and c".

Now we must change (x_2,x_1) to (y_2,y_1) in S_4. We get then :

For $z_0=0$,

$$S_4 \text{ maps } (0,0) \text{ onto } (0,1). \tag{21}$$

Finally, we must change z_0 into y_0 in S''. Here $y_0=z_0$. Thus :

For $(y_2,y_1)=(0,1)$,

$$c'' = 0. \tag{22}$$

Now we take $(y_2,y_1,y_0)=(0,1,1)$, which is paired to $(0,1,0)$. Its inverse image by f is $(x_2,x_1,x_0)=(0,1,1)$. We do the same procedure backwards. As $c''=0$ by (22), we get :

For $(y_2,y_1,y_0)=(0,1,1)$,

$$z_0 = 1. \tag{23}$$

Now (x_2,x_1) is transformed into (z_2,z_1) in S_4. Thus :

For $z_0 = 1$,

$$S_4 \text{ maps } (0,1) \text{ onto } (0,1). \tag{24}$$

Finally, S changes x_0 into z_0. Therefore :

For $(x_2,x_1)=(0,1)$,

$$c = 0. \tag{25}$$

Now take $(x_2,x_1,x_0)=(0,1,0)$, which is paired to $(0,1,1)$. It is mapped by f on $(1,0,0)$. By (25) we get :

For $(x_2,x_1,x_0)=(0,1,0)$,

$$z_0 = 0. \tag{26}$$

We pass now to S_4. We must get $(y_2,y_1) = (1,0)$. Thus :

For $z_0 = 0$,

$$S_4 \text{ maps } (0,1) \text{ onto } (1,0). \tag{27}$$

Finally S'' changes z_0 into y_0. As $z_0=y_0$, we have :

For $(y_2,y_1)=(1,0)$,

$$c'' = 0. \tag{28}$$

Now we take $(y_2,y_1,y_0)=(1,0,1)$ (paired to $(1,0,0)$), which is the image of $(0,0,1)$. By (28) we get

For $(y_2,y_1,y_0)=(1,0,1)$,

$$z_0 = 1. \tag{29}$$

Then in S_4 we get :

For $z_0 = 1$,

$$S_4 \text{ maps } (0,0) \text{ onto } (1,0). \tag{30}$$

Finally in S we have by (20)

For $(y_2,y_1,y_0)=(1,0,1)$

$$x_0 = 1, \tag{31}$$

which is what we had in the truth table. As $(x_2,x_1,x_0)=(0,0,0)$, the triple paired to $(0,0,1)$, has already been used, we have made a "loop". So we start again the procedure with an input which has not been used, say $(x_2,x_1,x_0)=(1,0,0)$.

We get then two loops of length 2 (the first one has length 4) and the procedure gives us the following facts :

For $(x_2,x_1)=(1,0)$,

$$c=0 \quad \text{(arbitrary).} \tag{32}$$

For $z_0=0$,

$$S_4 \text{ maps } (1,0) \text{ onto } (0,0). \tag{33}$$

For $(y_2,y_1)=(0,0)$,

$$c''=0. \tag{34}$$

For $(y_2,y_1,y_0)=(0,0,1)$,

$$z_0=1. \tag{35}$$

For $z_0=1$,

$$S_4 \text{ maps } (1,0) \text{ onto } (0,0). \tag{36}$$

For $(y_2,y_1,y_0)=(0,0,1)$,

$$x_0=1 \quad \text{(end of loop).} \tag{37}$$

Start now with $(x_2,x_1,x_0)=(1,1,0)$

For $(x_2,x_1)=(1,1)$,

$$c=0 \quad \text{(arbitrary).} \tag{38}$$

For $z_0=0$,

$$S_4 \text{ maps } (1,1) \text{ onto } (1,1). \tag{39}$$

For $(y_2,y_1)=(1,1)$,

$$c''=1. \tag{40}$$

For $(y_2,y_1,y_0)=(1,1,0)$,

$$z_0=1. \tag{41}$$

For $z_0=1$,

$$S_4 \text{ maps } (1,1) \text{ onto } (1,1). \tag{42}$$

For $(y_2,y_1,y_0)=(1,1,0)$,

$$x_0=1 \quad \text{(end of loop)}. \tag{43}$$

The looping procedure is now finished and have now all the informations on the setting of the CULM's. We get the following truth tables :

C : (44)

x_2	x_1	c
0	0	0
0	1	0
1	0	0
1	1	0

C'' : (45)

y_2	y_1	c''
0	0	0
0	1	0
1	0	0
1	1	1

$C' \rightarrow S_4$ (46)

z_0	x_2	x_1	y_2	y_1
0	0	0	0	1
0	0	1	1	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

Thus the setting of the control variables of C, C' and C'' can be determined.

In order to apply the looping algorithm, it is convenient to know the truth table of both f and its inverse. Generally a substitution f of size n is given in 2^n rows of n registers. If $[x_{n-1},\dots,x_0] = x$ $(0 \leqslant x < 2^n)$, then the bits in the row x are the n digits of $f(x_{n-1},\dots,x_0)$. In order to get f^{-1}, we apply the following algorithm :

We write $R(k,j)$ for the jth register in the kth row $(0 \leqslant j < n,$ $0 \leqslant k < 2^n)$.

We take 2^n other rows of n registers labelled $R'(k,j)$.

We make 2^n successive steps.

<u>Step i $(0 \leqslant i < 2^n)$</u>. Consider the ith row $R(i,0),\dots,R(i,n-1)$. If the contents of the registers form a number k $(0 \leqslant k < 2^n)$, then go to the kth row $R'(k,0),\dots,R'(k,n-1)$ and set their contents in order to form the number i.

<u>Example</u>. Take n=3 and the substitution (18). It is written in the following way in 24 registers

$$\begin{array}{ccc}
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \\
0 & 0 & 0 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
1 & 1 & 0
\end{array} \tag{45}$$

The 8 steps for filling the 24 new registers are illustrated below :

$$\begin{array}{ccc}
 & 0 & \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
0 & 0 & 0 \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot
\end{array}
\qquad
\begin{array}{ccc}
 & 1 & \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
0 & 0 & 0 \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
0 & 0 & 1 \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot
\end{array}
\qquad
\begin{array}{ccc}
 & 2 & \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
0 & 0 & 0 \\
\cdot & \cdot & \cdot \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot
\end{array}
\qquad
\begin{array}{ccc}
 & 3 & \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
0 & 0 & 0 \\
0 & 1 & 1 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot
\end{array}$$

$$\begin{array}{ccccccccccccc}
\multicolumn{3}{c}{4} & \multicolumn{3}{c}{5} & \multicolumn{3}{c}{6} & \multicolumn{3}{c}{7} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
\cdot & \cdot & \cdot & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 0 & 1 & 1 & 0
\end{array} \qquad (46)$$

Another interesting type of three stage network is when k=2 and t=1. Then the control algorithm can be derived from [13].

Both designs can be applied recursively. In the first case, one gets a network corresponding to the Benes network [11] for permutation networks.

A binary total substitution network of size 2 built by the three stage design is shown in Figure 15

C. General considerations

One can consider designs of not necessarily total substitution networks built by interconnecting substitution networks of smaller size.

In section IV of [11], ten operations on partial permutation networks are defined. These operations can also be done with substitution networks instead of partial permutation networks. This can be used for the building of substitution networks of relatively large size.

Two widely used operations are the union (i.e. setting of several substitution networks in parallel) and the composition (i.e. setting in series). The first one allows a cheap expansion of the substitution networks, while the second one increases the multiplicity (see for example the DES in [6]).

In [4] an algorithm is given for the expansion of a network *S* of size n to one of size n^g, where $g \geqslant 1$. In fact, this algorithm gives the network

$$\underbrace{(\ldots(S \times S) \times \ldots) \times S}_{g \text{ times}}, \qquad (47)$$

where × is the operation n° 6 defined in page 9 and illustrated in Figure 5 of [11]. We reproduce it on Figure 16.

In Section 5, we will analyse one particular property of the operation ×.

IV. Problems of cost and delay

We will compare the above-studied designs with respect to cost and delay in terms of logical gates of fan-in at most 2 and of fan-out 1 (NOT, AND, OR) in the binary case.

Suppose that these gates have the following respective costs and delays :

GATE	COST	DELAY
NOT	γ_0	δ_0
AND	γ	δ
OR	γ^1	δ^1

(48)

In MOSFET technology, we have :

$$\gamma = \gamma^1 = 3\gamma_0$$

and

$$\min\{\delta,\delta^1\} > \delta_0. \tag{49}$$

We can thus suppose that :

$$\gamma_0 \leqslant \min\{\gamma,\gamma^1\} \tag{50}$$

and

$$\delta_0 \leqslant \min\{\delta,\delta^1\}. \tag{51}$$

Let us look at the costs and delays of the components used in our designs.

(1) <u>The n to 2^n address decoder</u>

Its design using AND- and NOT-gates is shown in Figure 17 for n=3. As $\delta_0 \leqslant \delta$, we see easily that it has :

<u>cost</u> $\quad (2^{n+1}-4)\gamma + n\gamma_0 \approx 2^{n+1}\gamma.$ (52)

<u>delay</u> $\quad (n-1)\delta + \delta_0 \approx n\delta.$ (53)

(2) <u>The 2^n to n address encoder</u>

The input-output behavior of such network is incompletely defined. Therefore we will suppose that (7) holds.

Then such a network can be built by recursion. There are two recursion methods.

The first one is based on the fact that formula (7) can be split into two formulas :

$$z_{n-1} = \Big(\bigvee_{2^{n-1} \leqslant k < 2^{n-1}+2^{n-2}} v_k \Big) \vee \Big(\bigvee_{\substack{k \geqslant 2^{n-1} \\ k_{n-2}=1}} v_k \Big) \tag{54}$$

and

$$z_i = \Big(\bigvee_{\substack{k_i=1 \\ k < 2^{n-1}}} v_k \Big) \vee \Big(\bigvee_{\substack{k_i=1 \\ k \geqslant 2^{n-1}}} v_k \Big) \qquad (i=0,\dots,n-2). \tag{55}$$

The two terms in brackets in (55) are respectively the ith output of a 2^{n-1} to n-1 address encoder, with the inputs x_0 to $x_{2^{n-1}-1}$ in the first case, and $x_{2^{n-1}}$ to x_{2^n-1} in the second case.

One gets thus the design of Figure 18 illustrating the expansion from a 2^{n-1} to n-1 to a 2^n to n address encoder for n=4, and the following recursion formula :

$$\Gamma(n) = 2\Gamma(n-1) + (2^{n-2}+n-1)\gamma^1. \tag{56}$$

where $\Gamma(n)$ is the cost of a 2^n to n encoder. As a 2 to 1 address encoder is a simple connection between x_1 and z_0, we have :

$$\Gamma(1) = 0. \tag{57}$$

This gives an asymptotical cost of <u>at least</u> $2^{n+2}\gamma$.

For the delay, we get :

$$\Delta(n) = \Delta(n-1) + \delta^1 \tag{58}$$

and

$$\Delta(1) = 0, \tag{59}$$

in other words $\Delta(n) = (n-1)\delta^1$.

The second method has lower cost and equal delay. It is based on the following deomposition of formula (7) :

$$z_0 = \bigvee_{k \text{ odd}} v_k \tag{60}$$

and

$$z_{i+1} = \bigvee_{\substack{k_i=1 \\ k < 2^{n-1}}} (v_{2k} \vee v_{2k+1}) \qquad (i=0,\ldots,n-2). \tag{61}$$

Here (61) describes the input-output behavior of a 2^{n-1} to n-1 address encoder whose kth input $(0 \leqslant k < 2^n)$ is $v_{2k} \vee v_{2k+1}$. This gives the design of Figure 19 illustrating the expansion from a 2^{n-1} to n-1 to a 2^n to n address encoder for n=4, and the following recursion formula

$$\Gamma(n) = 2(2^{n-1}-1)\ \gamma^1 + \Gamma(n-1). \tag{62}$$

Using (57), we get by induction :

<u>cost</u> $\Gamma(n) = (2^{n+1}-2n-2)\ \gamma^1 \approx 2^{n+1}\gamma^1.$ (63)

The delay is given by :

$$\Delta(n) = \max\{(n-1)\gamma^1, \gamma^1 + \Delta(n-1)\}. \tag{64}$$

With (59), this gives :

<u>delay</u> $\Delta(n) = (n-1)\gamma^1 \approx n\gamma^1.$ (65)

Figure 20 shows a 16 to 4 address encoder built by applying recursively the decomposition of Figure 19.

(3) <u>The permutation network on m bits</u>

We suppose that we use Green's network (see [11], page 15). It has the following cost and delay :

$$(m \lceil \log_2(m) \rceil - 2^{\lceil \log_2(m) \rceil} + 1)\ \Gamma, \tag{66}$$

$$(2 \lceil \log_2(m) \rceil - 1)\Delta, \tag{67}$$

where Γ and Δ are the cost and the delay of a 2-cell and $\lceil u \rceil$ denotes the smallest integer v such that $v \geqslant u$. Note that if $m=2^n$ (as in our design), our formulas reduces to :

$$(2^n(n-1)+1)\Gamma \tag{68}$$

and

$$(2n-1)\Delta. \tag{69}$$

Now the design of a 2-cell is given in Figure 2.b of [11]. We get then :

$$\Gamma = \gamma_0 + 4\gamma + 2\gamma^1. \quad (70)$$

$$\Delta = \delta_0 + \delta + \delta^1. \quad (71)$$

Thus the permutation network on m bits has :

<u>cost</u> $\approx$ $m \log_2(m)\ (\gamma_0 + 4\gamma + 2\gamma^1)$. (72)

<u>delay</u> $\approx$ $2 \log_2(m)\ (\delta_0 + \delta + \delta^1)$. (73)

To this we must add the cost and delay of the <u>control algorithm</u>.

(4) <u>The enabler on m bits</u>.

As an OR-gate of fan-in m can be built with m-1 OR-gates of fan-in 2 in $\lceil \log_2(m) \rceil$ stages, it has :

<u>cost</u> $m\gamma + (m-1)\gamma^1 \approx m(\gamma + \gamma^1)$. (74)

<u>delay</u> $\delta + \lceil \log_2(m) \rceil\ \delta^1 \approx \log_2(m)\delta^1$. (75)

Now we can give the respective costs and delays of the designs described in Sections 2 and 3.

A. <u>CULM's</u>

(1) <u>The direct design</u>. It contains one n to 2^n address decoder and m copies of an enabler on 2^n bits. By (52), (53), (74) and (75), an n to m CULM has :

<u>cost</u> $((m+2)2^n - 4)\gamma + m(2^n - 1)\gamma^1 + n\gamma_0 \approx m2^n(\gamma + \gamma^1)$. (76)

$$\underline{\text{delay}} \qquad n(\delta+\delta^1)+\delta_0 \approx n(\delta+\delta^1). \tag{77}$$

Note that the control inputs have only the delay of the enabler, that is :

$$\delta + n\delta^1. \tag{78}$$

(2) <u>The expanded design</u>. Let us consider the design of Figure 10 in the binary case. Suppose that the t to m and the n to $m2^t$ CULM's used in this design are built with the direct design. By (76), we get the following cost :

$$(m(2^{n+t}+2^t)+2^{n+1}+2^{t+1}-8)\gamma$$

$$+ m(2^{t+n}-1)\gamma^1+(n+t)\gamma_0 \approx m2^{n+t}(\gamma+\gamma^1). \tag{79}$$

Now if we apply (76) for a n+t to m CULM, we get the following :

cost of the direct design - cost of the expanded one

$$= (2^{n+t+1}-2^{n+1}-2^{t+1}+4-m2^t)\gamma, \tag{80}$$

which is relatively small in relation to the total cost. This quantity is positive if and only if we have :

$$2(2^n-1)(2^t-1) > m2^t-2, \tag{81}$$

which is true whenever :

$$m \leqslant 2(2^n-1) \tag{82}$$

For substitution network, we have m=n+t and (80) becomes :

$$(2^{m+1} + 4 - \frac{2^{m+1}}{2^t} - (2+m)2^t)\gamma \tag{83}$$

By calculating the first and second derivatives of this formula with respect of $x=2^t$, we see that for a constant m, (83) reaches its maximum for

$$t = \frac{1}{2}(m+1 - \log_2(2+m)) \tag{84}$$

Thus the expanded network has lowest cost for this value of t, but it is asymptotically equivalent to the cost of the direct design.

We will now look at the delay of the expanded design.

a) control input signals : enabler on 2^n bits, enabler on 2^t bits

b) n first input signals : n to 2^n address decoder, enabler on 2^n bits, enabler on 2^t bits

c) t last input signals : t to 2^t address decoder, enabler on 2^t bits.

Using (53) and (75), we get the following three respective delays :

$$2\delta + (n+t)\delta^1, \tag{85}$$

$$(n+1)\delta + (n+t)\delta^1 + \delta_0, \tag{86}$$

and

$$t\delta + t\delta^1 + \delta_0. \tag{87}$$

Clearly (86) is larger than (85). Moreover (87) is larger than (86) if and only if

$$\delta^1 < \frac{t-n-1}{n}\,\delta, \tag{88}$$

which requires that $t > n+1$.

By (77), the direct design has delay :

$$(n+t)\delta + (n+t)\delta^1 + \delta_0, \tag{89}$$

which is larger than (87) and equal to (86) for t=1, but larger than it otherwise.

Thus the expanded design is cheaper in terms of delay.

B. Total substitution networks of size n.

(1) The classical direct design. It is built with one n to 2^n address decoder, one permutation network on 2^n bits and one 2^n to n address encoder. Using (52),(53),(63),(65),(68),(69),(70) and (71), we obtain the following

cost $\quad 2^{n+1}(2n-1)\gamma+2n(2^n-1)\gamma^1+(2^n(n-1)+n+1)\gamma_0 \approx n2^n(4\gamma+2\gamma^1+\gamma_0) \quad$ (90)

delay $\quad (3n-2)(\delta+\delta^1)+2n\delta_0 \approx n(3\delta+ 3\delta^1+2\delta_0) \quad$ (91)

To these figures must yet be added the cost and delay of the control algorithm of the permutation network. We see also that this design has more than twice the cost and delay than the design of a CULM.

(2) The expanded design. We will only give an approximation for its cost and delay. Consider the design of Figure 13. By Stirling's formula, we can consider that for any integer m we have

$$\log_2((2^m)!) \approx 2^m \log_2(2^m) = m2^m \tag{92}$$

Hence we can consider that in Figure 13 for k=2 we have :

$$u \approx n.2^n \tag{93}$$

and

$$v \approx t.2^t \tag{94}$$

We get the following approximative costs for the components

$$\mathcal{C}_t \; : \; \text{cost} \approx n.2^{n+t}(\gamma+\gamma^1), \tag{95}$$

$$\text{delay} \approx t(\delta+\delta^1), \tag{96}$$

$$\mathcal{C}_n \; : \; \text{cost} \approx t.2^{n+t}(\gamma+\gamma^1), \tag{97}$$

$$\text{delay} \approx n(\delta+\delta^1), \tag{98}$$

$$\mathcal{S}_t \; : \; \text{cost} \approx t.2^t(\lambda\gamma+\lambda^1\gamma^1+\lambda_0\gamma_0), \tag{99}$$

$$\text{delay} \approx t(\mu\delta+\mu^1\delta^1+\mu_0\delta_0), \tag{100}$$

$$\mathcal{S}_n \; : \; \text{cost} \approx n2^n(\lambda\gamma+\lambda^1\gamma^1+\lambda_0\gamma_0), \tag{101}$$

$$\text{delay} \approx n(\mu\gamma+\mu^1\gamma^1+\mu_0\delta_0), \tag{102}$$

where we have inaccordance to (76),(77),(90),(91) :

$$1 \leqslant \lambda \leqslant 4, \tag{103}$$

$$1 \leqslant \lambda^1 \leqslant 2, \tag{104}$$

$$0 \leqslant \lambda_0 \leqslant 1, \tag{105}$$

$$1 \leqslant \mu \leqslant 3, \tag{106}$$

$$1 \leqslant \mu^1 \leqslant 3, \tag{107}$$

and

$$0 \leqslant \mu_0 \leqslant 2, \tag{108}$$

since we suppose that the design of $\mathcal{S}_n$ and $\mathcal{S}_t$ may be anything having cost and delay between those of a CULM and those of the classical direct design. We get then the following :

$$\underline{\text{cost}} \approx (2n+t)2^{n+t}(\gamma+\gamma^1)+(2n2^n+t2^t)(\lambda\gamma+\lambda^1\gamma^1+\lambda_0\gamma_0) \tag{109}$$

$$\underline{\text{delay}} \approx (2t+n)(\delta+\delta^1)+(2n+t)(\mu\delta+\mu^1\delta^1+\mu_0\delta_0) \tag{110}$$

This design is more expensive than the CULM in terms of cost and delay. It has higher delay than the classical direct design, but its cost may be lower given a suitable choice of n and t.

(3) <u>Recursive iteration of the three stage decomposition with n=1.</u>

We suppose that we build for every integer $m > 0$ the total substitution network $S(m)$ defined recursively as follows :

- $S(1)$ is the modulo 2 adder of fan-in 2
- For $m > 1$, $S(m)$ is obtained from $S(m-1)$ and $S(1)$ by the three stage design as shown in Figure 14 for m=3.

Let $\kappa(m)$ be the number of control inputs of $S(m)$. We have :

$$\kappa(1) = 1 \tag{111}$$

and for $m > 1$:

$$\kappa(m) = 2^{m-1} + 2\kappa(m-1) + 2^{m-1}, \tag{112}$$

where each summand is equal to the number of control inputs of C, C' and C'' (cfr. Figure 14). By induction, we obtain

$$\kappa(m) = m2^m - 2^{m-1}, \tag{113}$$

which is also the cost of the Benes network in 2-cells (see [11]). This is no coincidence, since the construction of $S(m)$ corresponds to the construction of the Benes network for permutation networks.

Let γ_2 and δ_2 be respectively the cost and delay of the modulo 2 adder.

Now for $m > 1$ the design of $S(m)$ contains the following components (see Figure 14) :

- Two modulo 2 adders of

$$\text{cost} : \gamma_2 \tag{114}$$

$$\text{delay} : \delta_2 \tag{115}$$

- Two $m-1$ to 1 CULM's of

$$\text{cost} : (3\times 2^{m-1}-4)\gamma+(2^{m-1}-1)\gamma^1+(m-1)\gamma_0 \tag{116}$$

$$\text{delay} : (m-1)(\delta+\delta^1)+\delta_0 \tag{117}$$

- One copy of S_{m-1}
- One 1 to $\kappa(m-1)$ CULM of

$$\text{cost} : 2\kappa(m-1)\gamma+\kappa(m-1)\gamma^1+\gamma_0 \tag{118}$$

$$\text{delay} : \delta+\delta^1+\delta_0 \tag{119}$$

Let us write $\Gamma(m)$ and $\Delta(m)$ for the cost and delay of $S(m)$. We have thus :

$$\Gamma(1) = \gamma_2 \tag{120}$$

$$\Delta(1) = \delta_2 \tag{121}$$

and for $m > 1$:

$$\Gamma(m) = 2\gamma_2+2(3\times 2^{m-1}-4)\gamma+2(2^{m-1}-1)\gamma^1+2(m-1)\gamma_0+\Gamma(m-1)+2\kappa(m-1)\gamma+\kappa(m-1)\gamma^1+\gamma_0$$

$$= \Gamma(m-1)+2\gamma_2+(2m-1)\gamma_0+(2^m-2+(m-1)2^{m-1}-2^{m-2})\gamma^1+(2^{m+1}+2^m-8+(m-1)2^m-2^{m-1})\gamma$$

$$= \Gamma(m-1)+2\gamma_2+(2m-1)\gamma_0+((2m+1)2^{m-2}-2)\gamma^1+((2m+3)2^{m-1}-8)\gamma \tag{122}$$

By induction we get for every $m \geqslant 1$

$$\underline{\text{cost}} \quad \Gamma(m) = (2m-1)\gamma_2+(m^2-1)\gamma_0$$

$$+ ((2m-1)2^{m-1}-2(m-1)-1)\gamma^1$$

$$+ ((2m+1)2^m-8(m-1)-6)\gamma$$

$$\approx m2^m(\gamma^1+2\gamma) \qquad (123)$$

Now we have for $m > 1$:

$$\Delta(m) = 2\delta_2+2(m-1)(\delta+\delta^1)+2\delta_0+\Delta(m-1)+\delta+\delta^1+\delta_0$$

$$= \Delta_{m-1}+2\delta_2+2\delta_0+m(\delta+\delta^1) \qquad (124)$$

By induction, this gives :

$$\underline{\text{delay}} \quad \Delta(m) = (2m-1)\delta_2+3(m-1)\delta_0+ \frac{m(m-1)}{2}(\delta+\delta^1) \approx m^2(\delta+\delta^1) \qquad (125)$$

In order to make the final approximation, we used the fact that γ_2 and δ_2 are of the same order respectively as $(\gamma+\gamma^1)$ and $(\delta+\delta^1)$. This follows from (50),(51) and from the design of a modulo 2 adder using the 3 conventional gates shown in Figure 21.

We see that the cost of S_m is comparable (less than twice) to the cost of a CULM. However its delay is extremely high : it is in m^2 instead of m.

<u>Conclusion</u>. CULM's are the easiest and cheapest way to generate all substitutions of size n. However, they cannot be properly considered as substitution networks. For total substitution networks, the classical direct design has lowest delay and highest cost, while the design $S(n)$ has lowest cost and very high delay.

V. Variable dependency in substitutions

Consider a substitution $f : K^n \to K^n$. The domain of f is written with variables $x_0,\dots,x_{n-1}$ and the image with $y_0,\dots,y_{n-1}$. We wish to describe the dependency relations between the variables y_i and x_j $(i,j \in \{0,1,\dots,n-1\})$.

Suppose that we transform a plaintext $x_0,\dots,x_{n-1}$ into a cyphertext $y_0,\dots,y_{n-1}$ and that for some $i \in \{0,\dots,n-1\}$ there is a relatively small number t such that

$$y_i = g(x_{j_0},\dots,x_{j_{t-1}}) \tag{126}$$

with $j_0,\dots,j_{t-1} \in \{0,1,\dots,n-1\}$. Then given several samples of plaintexts, it is relatively easy to find the relation between y_i and the plaintext variables. To avoid this situation, it is good to require that the variable y_i depends of all the variables $x_0,\dots,x_{n-1}$.

Let us make the following four definitions :

Let y be a function of n variables $x_0,x_1,\dots,x_{n-1}$: $y = y(x_0,\dots,x_{n-1})$. Then :

(i) The <u>strong dependency rate</u> $D(y/x_i)$ of y on x_i $(0 \leqslant i \leqslant n-1)$ is $1/|K|^{n-1}$ times the number of (n-1)-tuples $(x_0,\dots,x_{i-1},x_{i+1},\dots,x_{n-1})$ such that for any a, a' $\in$ K, a $\neq$ a' implies that $g(x_0,\dots,x_{i-1},a,x_{i+1},\dots,x_{n-1}) \neq g(x_0,\dots,x_{i-1},a',x_{i+1},\dots,x_{n-1})$.

(ii) The <u>weak dependency rate</u> $d(y/x_i)$ of y on x_i $(0 \leqslant i \leqslant n-1)$ is $1/|K|^{n-1}$ times the number of (n-1)-tuples $(x_0,\dots,x_{i-1},x_{i+1},\dots,x_{n-1})$ such that there exist some a, a' $\in$ K with $g(x_0,\dots,x_{i-1},a,x_{i+1},\dots,x_{n-1}) \neq g(x_0,\dots,x_{i-1},a', x_{i+1},\dots,x_{n-1})$.

(iii) y is <u>strongly dependent</u> on x_i if $D(y/x_i) \neq 0$.

(iv) y is <u>weakly dependent</u> on x_i if $d(y/x_i) \neq 0$.

We can make the following immediate remarks :

(a) $$0 \leqslant D(y/x_i) \leqslant d(y/x_i) \leqslant 1 \tag{127}$$

(b) (iii) implies (iv) (by (a)).

(c) When $|K| = 2$, the weak dependency rate is equal to the strong dependency rate. In this case we will speak of <u>dependency rate</u> and <u>dependency</u>.

In [4], the authors define dependency in the binary case and define a number relatively similar to the dependency rate (see the number $Q_{i,j}$ on page 751 of [4]).

Consider again the substitution f defined above. Generalizing [4], we will say that f is <u>weakly complete</u> if every y_j is weakly dependent on x_i, and <u>strongly complete</u> if every y_j is strongly dependent on x_i. In the binary case, we will say that f is <u>complete</u>.

Let us make a few remarks :

(a) If f is affine (with K equal to a field) and if we write

$$y_j = \sum_{i=0}^{n-1} x_i f_{ij} + f_j \quad (j=0,1,\dots,n-1), \tag{128}$$

where det $(f_{ij}) \neq 0$, then for any $i,j \in \{0,1,\dots,n-1\}$ we have either :

$$\text{(i) } f_{ij} \neq 0 \text{ and } d(y_j/x_i) = D(y_j/x_i) = 1 \tag{129}$$

$$\text{(ii) } f_{ij} = 0 \text{ and } d(y_j/x_i) = D(y_j/x_i) = 0 \tag{130}$$

(b) It follows that if f is affine, then f is complete if and only if $f_{ij} \neq 0$ for all $i,j \in \{0,1,\ldots,n-1\}$. As $\det(f_{ij}) \neq 0$, this is impossible if $|K| = 2$ and $n \geqslant 2$ (this is Theorem 4 of [4]).

(c) If K=GF(2) and $d(y_j/x_i)=1$, then y_j is linear in x_i. It follows from (b) that if K=GF(2) and $d(y_j/x_i)=1$ for any $i,j \in \{0,1,\ldots,n-1\}$, then n=1.

Now let m and n be two positive integers and consider m substitution networks $S_0,\ldots,S_{m-1}$ on n bits realizing the substitutions $f_0,\ldots,f_{m-1}$ respectively, and n substitution networks $T_0,\ldots,T_{n-1}$ on m bits realizing the substitutions $g_0,\ldots,g_{n-1}$ respectively. Connect them as in Figure 16, i.e. connect the output j of S_i ($j=0,\ldots,n-1$; $i=0,\ldots,m-1$) to the input i of T_j. Consider the resulting network N and the substitution h that it realizes.

We now make the following convention : Given a substitution $\gamma : (\xi_0,\ldots,\xi_{\mu-1}) \to (\zeta_0,\ldots,\zeta_{\mu-1})$, we will write $D_\gamma(j/i)$ for $D(\zeta_j/\xi_i)$.

Now for any $s,i \in \{0,1,\ldots,m-1\}$ and $r,j \in \{0,1,\ldots,n-1\}$, we have :

$$D_h(rm+s/in+j) = D_{f_i}(r/j).D_{g_r}(s/i) \tag{131}$$

Indeed, suppose that the input signal in+j of N runs over K and that the other input signals of N remain fixed. Then the output signals of the networks $S_{i'}$ ($i' \neq i$) remain fixed. It follows that the input signals i' ($i' \neq i$) of T_r are fixed. Therefore the output signal rm+s of N runs over K if and only if : (1) the input signal i of T_r (i.e. the output signal r of S_i) runs over K;

(2) the input signals i' ($i' \neq i$) of T_r form a (m-1)-tuple having the property described in definition (i);

(3) the input signals j' ($j' \neq j$) of S_i form a (n-1)-tuple having that same property.

The number of choices of (m-1)-tuples in (2) is $|K|^{m-1}\ D_{g_r}\ (s/i)$.

Given a fixed choice for such a (m-1)-tuple, then the output signal r of each $S_{i'}$ $(i'\neq i)$ will be fixed. Hence there will be $|K|^{n-1}$ choices for the input signals of $S_{i'}$ for each such i'.

Finally, the number of choices of (n-1)-tuples in (3) is $|K|^{n-1}$ $D_{f_i}\ (r/j)$.

Thus the total number of choices of (mn-1)-tuples of input signals $\neq$ in +j will be

$$|K|^{m-1}.D_{g_r}(s/i).(|K|^{n-1})^{m-1}.|K|^{n-1}.D_{f_i}(r/j)$$

$$= |K|^{nm-1}.D_{f_i}(r/j).D_{g_r}(s/i)$$

By definition of D_h (rm+s/in+j), (131) follows.

This result is illustrated in Figure 22 for m=n=4, i=2, j=3, r=0, s=1.

Let us give some applications of this result :

First suppose that all S_i's are equal to a substitution network S realizing f and that all T_j's are equal to a substitution network T realizing g. Then the network becomes $S \times T$ and formula (131) becomes :

$$D_h(rm+s/in+j) = D_f(r/j).D_g(s/i) \qquad (132)$$

Now for any substitution γ of length μ, we put

$$D_\gamma = \min\{D_\gamma(j/i)\,|\,i,j \in \{0,1,\ldots, -1\}\} \qquad (133)$$

D_γ is called the strong dependency rate of γ. It is easy to see that (132) implies the following :

$$D_h = D_f.D_g \tag{134}$$

Now, for a substitution γ, γ is strongly complete if and only if $D_\gamma \neq 0$. Thus (134) implies that if f and g are strongly complete, then h is.

Applying this to the network (47), we get Theorems 1 and 2 of [4].

This property justifies the interest of the operation $\times$.

Examples of complete substitutions are given in [4] for the binary case. For $|K| > 2$, if $|K|$ is a prime power q, then we can take $K = GF(q)$, the field of q elements. Then an affine substitution whose matrix has only non-zero entries (see (128)) is strongly complete. If $\lambda = |K|$ is not a prime power, then we can try to take a substitution of type (128) over the ring Z_λ of integers modulo λ, and the condition for f to be a substitution is that $\det(f_{ij})$ has an inverse in Z_λ for multiplication. Here (129) and (130) become :

(i) If $f_{ij} = 0$, then $d(y_j/x_i)=D(y_j/x_i)=0$ (135)

(ii) If $f_{ij}\neq 0$ but f_{ij} has no inverse in Z_λ, then $d(y_j/x_i)=1$ and $D(y_j/x_i)=0$ (136)

(iii) If f_{ij} has an inverse in Z_λ, then $d(y_j/x_i)=D(y_j/x_i)=1$ (137)

Thus one can try to find a matrix with inversible determinant whose entries are also inversible.

Another method is to decompose λ in factors. Suppose that $\lambda = \lambda_1.\lambda_2$ and take K_1 and K_2 of respective size λ_1 and λ_2. Then there is a bijection :

$$\phi : K \to K_1 \times K_2 : y \to (y',y'') \tag{138}$$

Now we define the natural bijection :

$$\psi : (K_1 \times K_2)^n \to K_1^n \times K_2^n$$

$$((y'_{n-1},y''_{n-1}),\dots,(y'_0,y''_0)) \to ((y'_{n-1},\dots,y'_0),(y''_{n-1},\dots,y''_0) \tag{139}$$

Now given two substitutions $f_1 : K_1^n \to K_1^n$ and $f_2 : K_2^n \to K_2^n$, then let (f_1,f_2) be the corresponding substitution of $K_1^n \times K_2^n$:

$$(f_1,f_2):(\underline{\underline{y}}',\underline{\underline{y}}'') \to (\underline{\underline{y}}'\ f_1,\underline{\underline{y}}''\ f_2) \tag{140}$$

Then we can define a substitution $f : K^n \to K^n$ by :

$$f = \phi.\psi.(f_1,f_2).\psi^{-1}.\phi^{-1} \tag{141}$$

We write then :

$$f = (f_1,f_2)_\phi \tag{142}$$

It is easily checked that for any $i,j \in \{0,\dots,n-1\}$ we have :

$$D_f(i/j) = D_{f_1}(i/j).D_{f_2}(i/j) \tag{143}$$

Thus, if both f_1 and f_2 are strongly complete, then f is.

Note that strongly complete substitutions of large size can be built with the product operation $\times$, because of (132).

Appendix A

Changes in size or in alphabet.

We can suppose that in order to encypher a plaintext of length n in a given alphabet K, we get a cyphertext of larger length n', or in a larger alphabet K', or both.

We can thus define a pseudo-substitution as an injection

$$f : K^n \to K'^{n'} , \qquad (144)$$

where $$|K'|^{n'} \geqslant |K|^n . \qquad (145)$$

(An injection is a map such that two distinct element of the domain have distinct images).

Note that if both K and K' can be expressed as powers of a set K_0, then the problem reduces to a change in length.

Suppose that we write in the binary alphabet $K = \{0,1\}$. Typing characters can be represented as elements of K^7. So a usual text of length n will be written as a text of length 7n in K.

Suppose that we want to make a pseudo substitution $f: K^{7n} \to K^{mn}$, with $m > 7$. One idea is to divide the text in n blocks in K^7 and to apply to the ith block $(i=0,\ldots,n-1)$ a pseudosubstitution $f_i : K^7 \to K^m$, and to take the combination of all these pseudosubstitutions. Afterward, we can apply to the resulting text a substitution of length mn. Let us give an example.

Take m=7+d. Suppose that there is some number μ such that for any typographical sign $*$, the relative frequency $\phi(*)$ of $*$ satisfies the following approximation :

$$\phi(*) \overset{\sim}{\sim} \lambda(*)/\mu \quad \text{for some} \quad \lambda \in \{0,\dots,2^d-1\} .$$

Then we can take f_i to be :

$$f_i : (x_6,\dots,x_0) \to (x_6,\dots,x_0,\ y_{d-1},\dots,y_0) , \tag{146}$$

where for $(x_6,\dots,x_0)$ corresponding to $*$,

$$[\ y_{d-1},\dots,y_0\ ] \equiv v \pmod{\phi(*)} , \tag{147}$$

where $(x_6,\dots,x_0)$ appears for the vth time, (starting with $v=0$). (148)

We take the following example with $n=30$ and $d=2$. We restrict ourselves to texts in capital Latin letters and normal punctuation signs like , ; . : ? ! ( ) etc... (Then we can of course take K^6 instead of K^7).

We take the following text of length 30 (blanks included) :

DRINK MORE BEER IN THE SUMMER . (149)

We take the following values for $\lambda(*)$

3 for blank, E,T,O,A,N.

2 for I,R,S,H,D.

1 for L,C,F,M,U,P.

0 for the rest (150)

We get then the following text written in K^9 instead of K^7, where we write $*_i$ for $(*,i)$

DRINK MORE BEER IN THE SUMMER . (152)
000000001010120211300300000010

Afterwards we can apply to this text any substitution in K^{180}.

This idea can be applied to any number λ instead of 7. Thus, for any alphabets K,L, for any integers λ,μ,n such that ,

$$|K|^{\lambda} \leqslant |L|^{\mu} \tag{152}$$

a pseudosubstitution $K^{\lambda n} \to L^{\mu n}$ can be built in the following way. Take n pseudosubstitutions :

$$f_i\ K^{\lambda} \to L^{\mu} \qquad (i=0,\ldots,n-1) \tag{153}$$

and a substitution

$$g : L^{\mu n} \to L^{\mu n}$$

Then the map

$$(f_0 \cup \ldots \cup f_{n-1})g \tag{154}$$

is a pseudosubsitution $K^{\lambda n} \to L^{\mu n}$. (Here $\cup$ is the operation which consists in setting two substitutions in parallel on two neighbouring blocks of bits).

This contruction is illustrated in Figure 23 for $\lambda=3$, $\mu=4$ and n=5.

This construction sets the pattern for any possible pseudosubstitution. Take any alphabets K,L and any positive integers n,m such that $|K|^n \leqslant |L|^m$. If we write

$$n = n_0 + \ldots + n_{k-1} \tag{155}$$

$$\text{and} \qquad m = m_0 + \ldots + m_{k-1} \quad ,$$

where $k \geqslant 2$, $n_i \neq 0 \neq m_i$ for $i=0,\ldots,k-1$ and :

$$|K|^{n_i} \leqslant |L|^{m_i} \quad \text{for } i=0,\ldots,k-1, \tag{156}$$

then we can take k arbitrary substitutions

$$f_i : K^{n_i} \to L^{m_i} \qquad (i=0,\ldots,k-1) \quad . \tag{157}$$

It is easy to see then that every pseudosubstitution $f:K^n \to L^m$ can be expressed as

$$f = (f_0 \cup \ldots \cup f_{k-1})g \;, \tag{158}$$

where g is a substitution $L^m \to L^m$. It is sufficient to define

$$\underline{\underline{y}}g = \underline{\underline{y}}(f_0 \cup \ldots \cup f_{k-1})^{-1} f \tag{159}$$

when $\underline{\underline{y}}$ is in the image of $f_0 \cup \ldots \cup f_{k-1}$, and to complete g in an arbitrary bijective way for other values of $\underline{\underline{y}}$.

It may be interesting to choose f in such a way that the image of f forms an error-correcting code for the distance.

Appendix B.

The three stage design, the DES and non autonomous FSR's.

Consider the three stage design when $|K| = 2$ and $t=n$. If we draw a double line for a set of n connections, then Figure 13 is equivalent to Figure 24 It contains three stages of the network shown in Figure 25.

Now if we replace S_n by a bit by bit modulo 2 adder, which is a non-total substitution network, then we get (up to a left-right symmetry) the basic transformation used in the DES algorithm [6] .

This fact was first remarked by M. Davio.

Now a sequence of transformations of type 25 can be obtained by a non autonomous feedback shift register on $GF(2^n)$. Indeed, $GF(2^n)$ can be identified with K^n. Then the nonautonomous feedback shift register shown in Figure 26 performs at each chock pulse the transformation of Figure 25.

It is known that feedback shift registers can be used to generate pseudorandom sequences. This indicates a possible reason for the use of transformations like the one of Figure 25.

Acknowledgement. The author is endebted to M. Davio for useful suggestions and to J.-M. Goethals for references.

REFERENCES.

[1] S. Andresen, "The Looping Algorithm Extended to Base 2^t Rearrangeable Switching Networks", IEEE Trans. on Communications, vol. COM-25, n°10, 1057-1063, 1977.

[2] H. Feistel, "Cryptographs and Computer Privacy", Sci. Amer., pp. 15-23, May 1973.

[3] H. Feistel, W. A. Notz, J. L. Smith, "Some Cryptographic Techniques for Machine-to-Machine Data Communications", Proc. IEEE, vol. 63, n°11, 1545-1554, 1975.

[4] J. B. Kam, G. I. Davida, "Structured Design of Substitution-Permutation Encryption Networks", IEEE Trans. on Computers, vol. C-28, n°10, 747-753, 1979.

[5] A. G. Konheim, "Cryptographic Methods for Data Protection", in "Information Linkage between Applied Mathematics and Industry", ed. Peter C. C. Wang, Academic Press, pp. 137-157, 1979.

[6] R. Morris, N. J. A. Sloane, D. A. Wyner, "Assessment of the National Bureau of Standards Proposed Federal Data Encryption Standard", Cryptologia, vol. 1, 281-306, 1977.

[7] V. I. Neimann, "Structure et Commande Optimales de Réseaux de Connexion sans Blocage", Annales des Télécommunications, 232-238, 1969.

[8] H. R. Ramanujam, "Decomposition of Permutation Networks", IEEE Trans. on Computers, vol. C-22, n°7, 639-643, 1973.

[9] N. T. Tsao-Wu, "On Neisman's Algorithm for the Control of Rearrangeable Switching Networks", IEEE Trans. on Communications, vol. COM-22, n°6, 737-742, 1974.

[10] A. Waksman, "A permutation network", J.A.C.M., vol. 15, 159-163, 340, 1968.

[11] C. Ronse, "Cellular Permutation Networks : A Survey", Report R415, MBLE Research Laboratory, December 1979.

[12] M. Davio, J.P. Deschamps, A. Thayse, "Discrete and switching functions", McGraw-Hill, 1978.

[13] N. T. Tsao-Wu, D.C. Opferman, "On Permutation Algorithms for Rearrangeable Switching Networks", IEEE Int. Conf. Comm., Conf. Records, pp. 10/29-10/34, 1969.

[14] D. C. Opferman, N. T. Tsao-Wu, "On a Class of Rearrangeable Switching Networks. Part I : Control Algorithm", Bell System Tech. J., vol. 50, n° 5, 1579-1600, 1971.

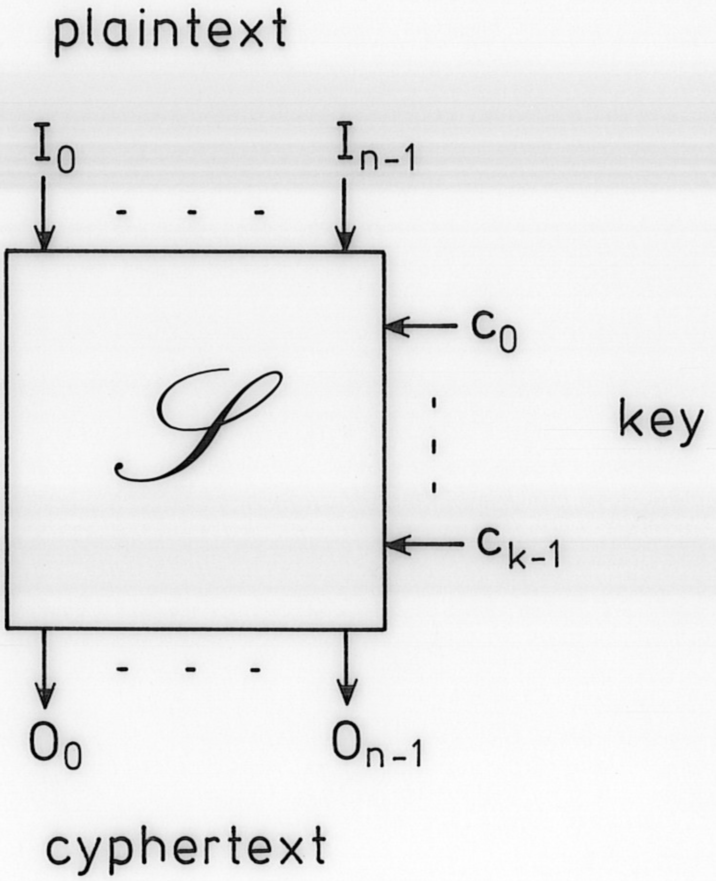
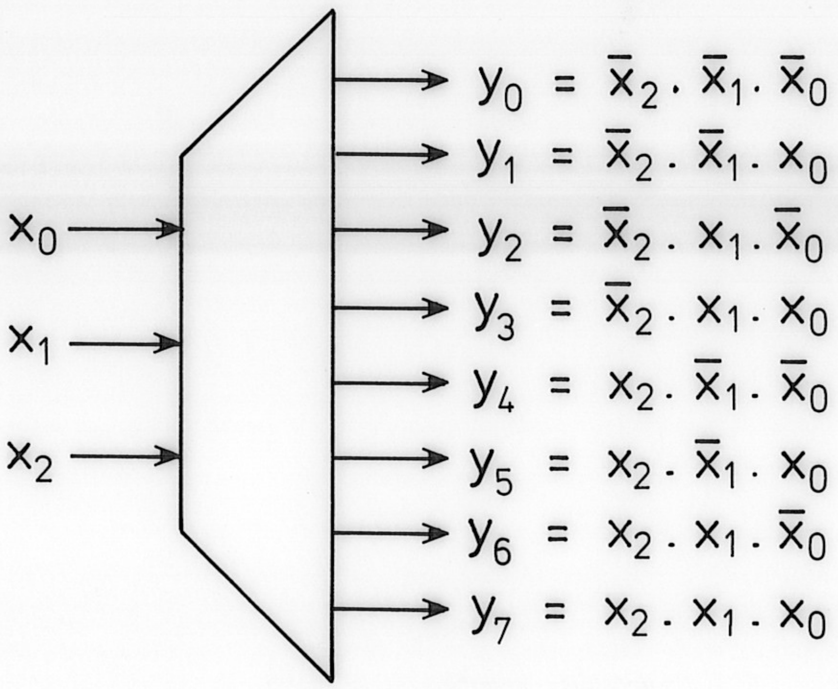
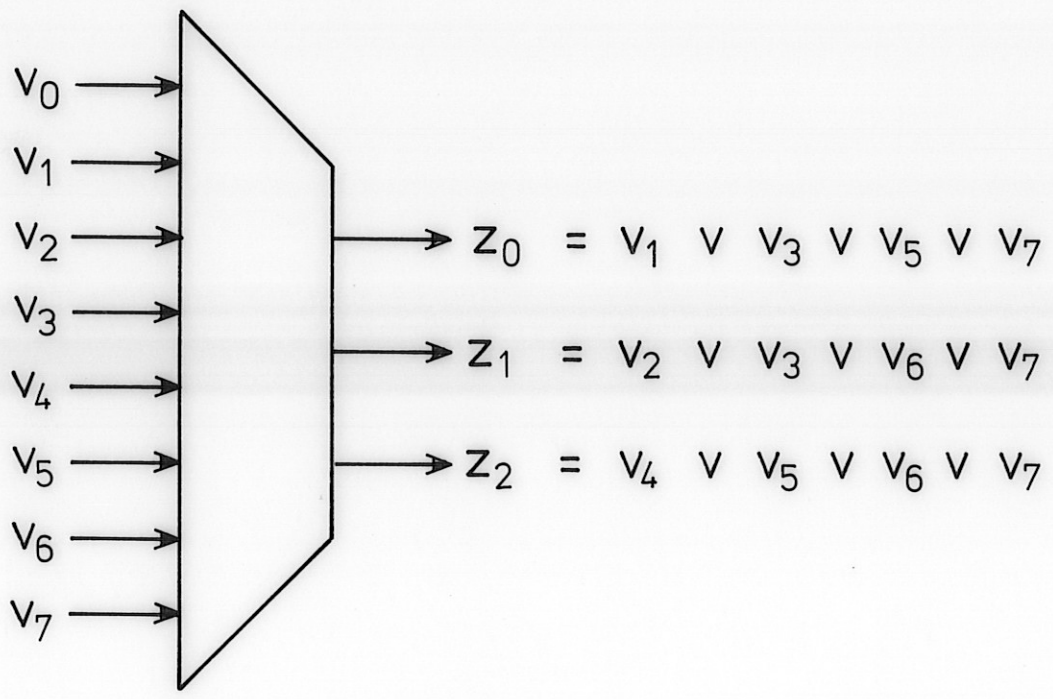


Figure 1.


x_0
x_1
x_2
$y_0 = \bar{x}_2 . \bar{x}_1 . \bar{x}_0$
$y_1 = \bar{x}_2 . \bar{x}_1 . x_0$
$y_2 = \bar{x}_2 . x_1 . \bar{x}_0$
$y_3 = \bar{x}_2 . x_1 . x_0$
$y_4 = x_2 . \bar{x}_1 . \bar{x}_0$
$y_5 = x_2 . \bar{x}_1 . x_0$
$y_6 = x_2 . x_1 . \bar{x}_0$
$y_7 = x_2 . x_1 . x_0$

<u>Figure 2.</u>

A 3 to 8 address decoder.



v_0
v_1
v_2
v_3
v_4
v_5
v_6
v_7
$z_0 = v_1 \vee v_3 \vee v_5 \vee v_7$
$z_1 = v_2 \vee v_3 \vee v_6 \vee v_7$
$z_2 = v_4 \vee v_5 \vee v_6 \vee v_7$

Figure 3.

An 8 to 3 address encoder.

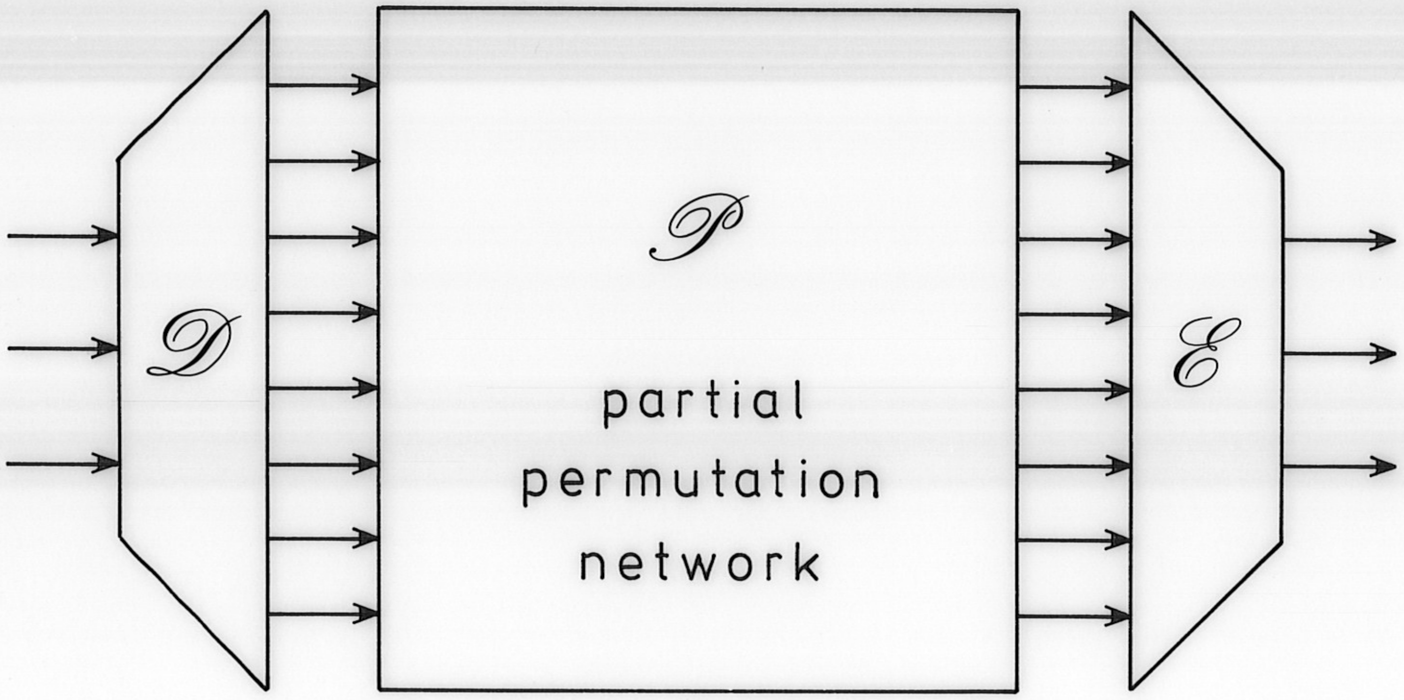


Figure 4.

A substitution network of size 3.

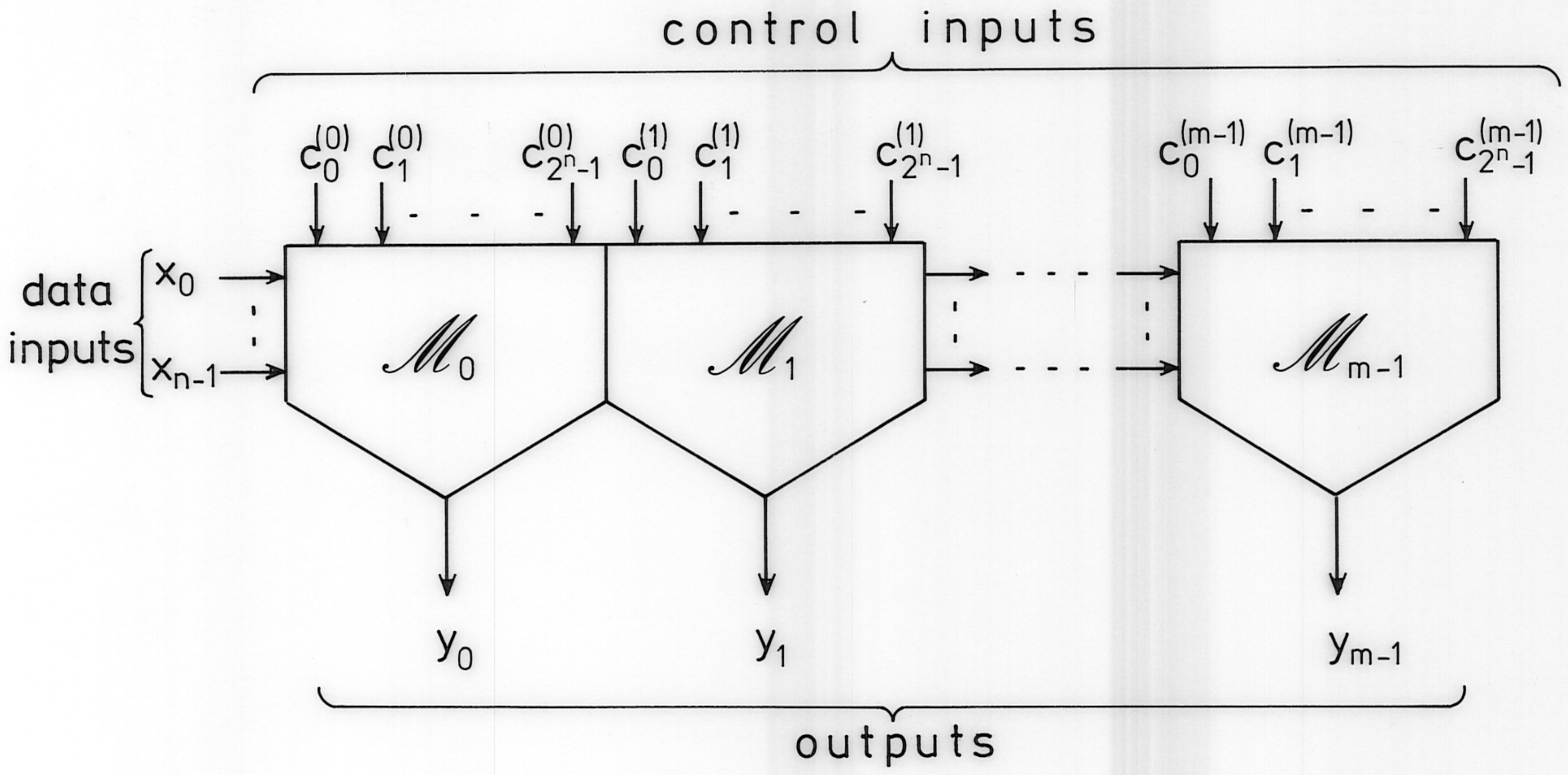


Figure 5.

An n to m CULM.

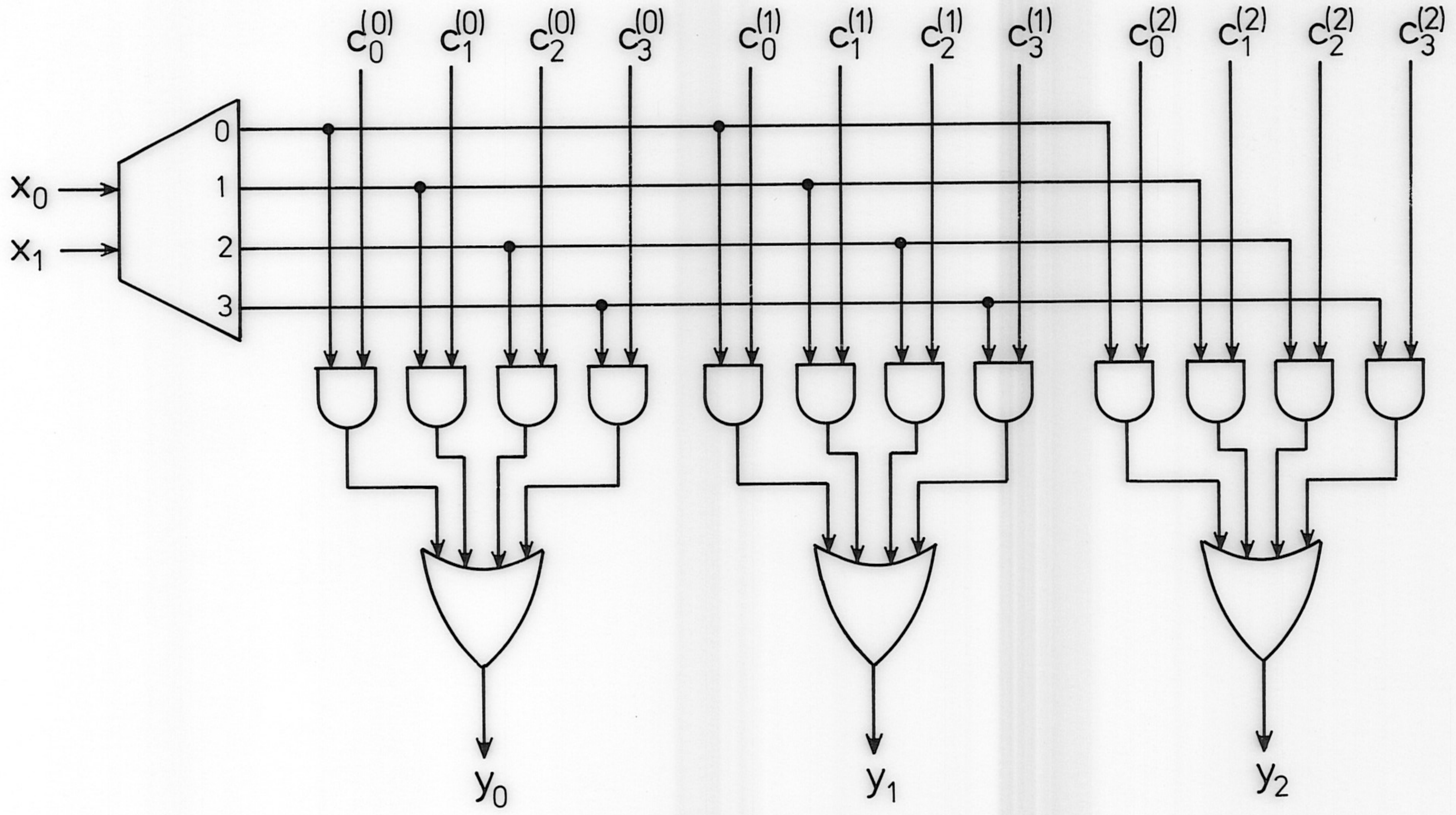


Figure 6. A 2 to 3 CULM.

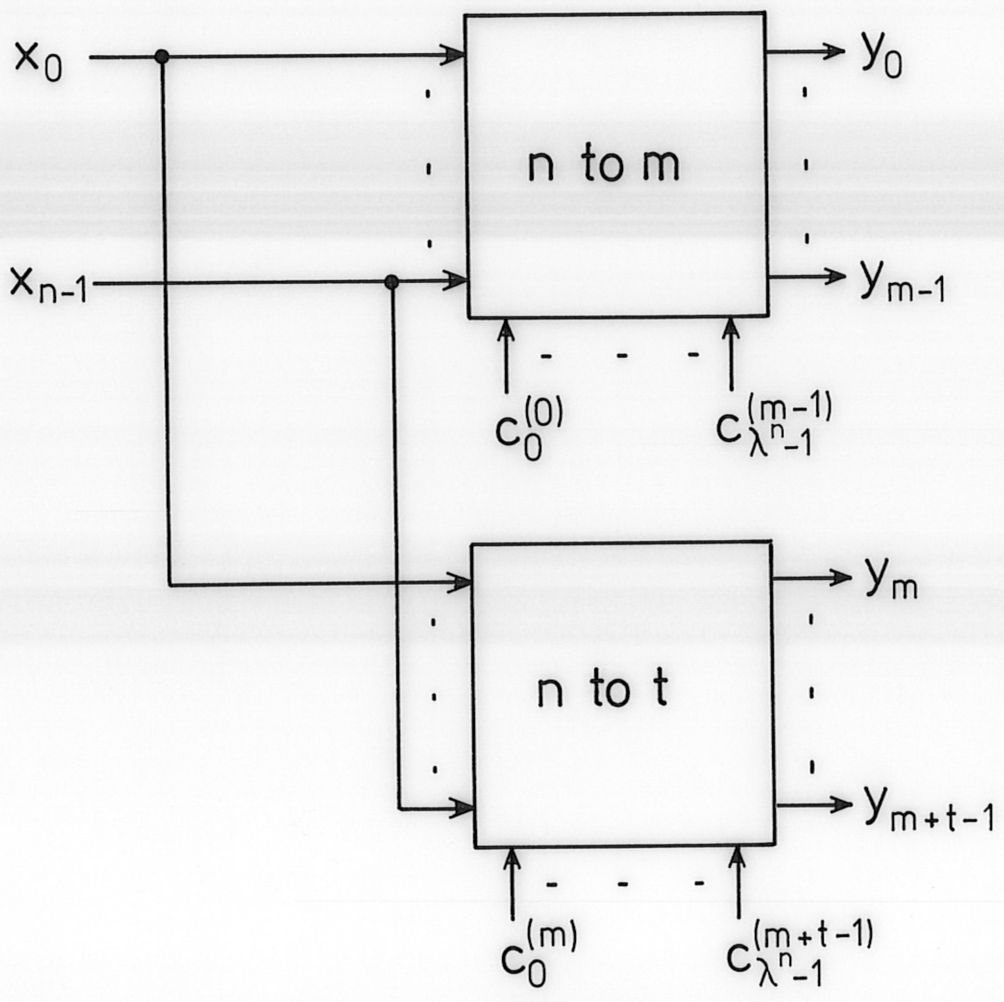


Figure 7.

An n to (m+t) CULM.

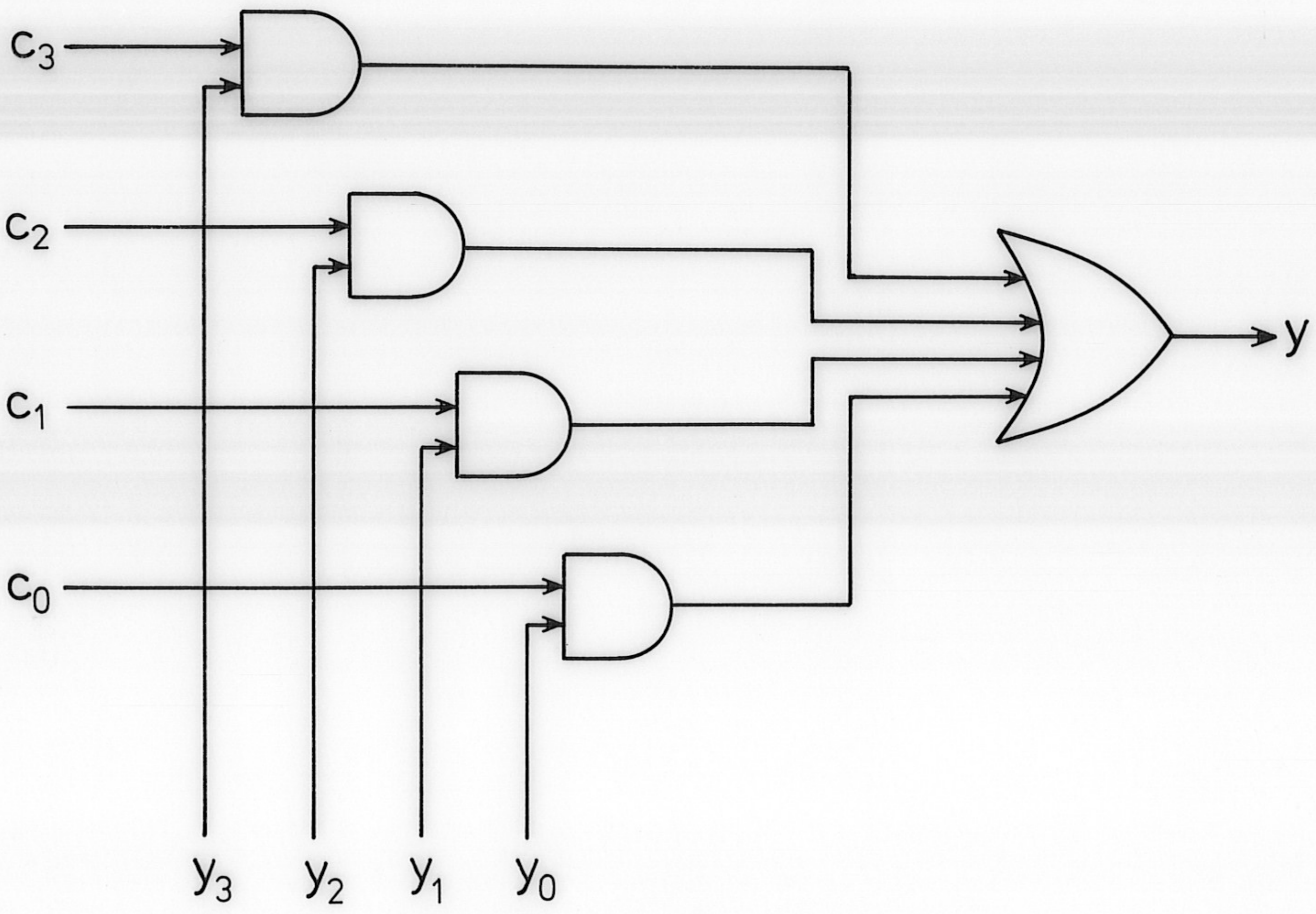


Figure 8. An enabler on 4 bits.

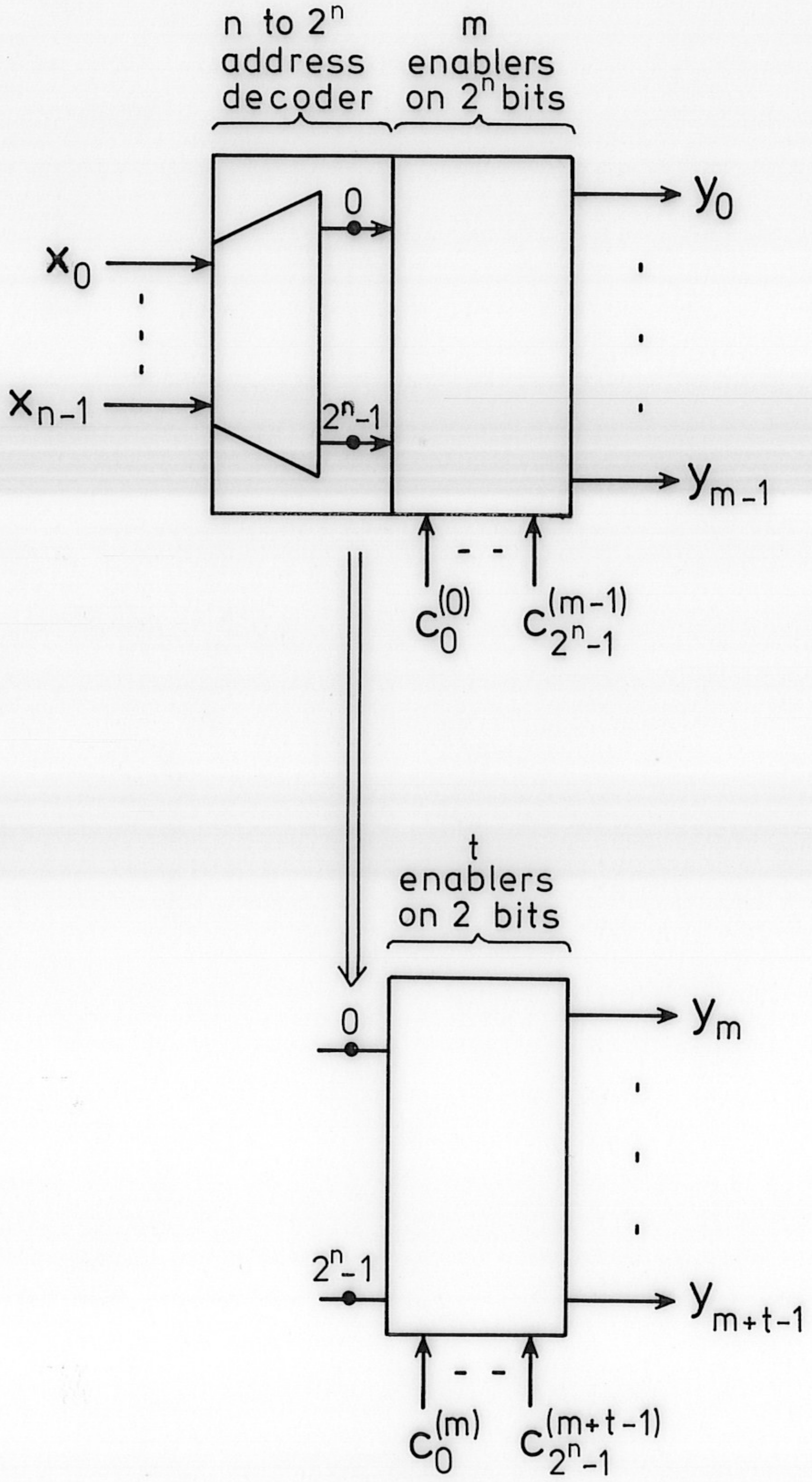


Figure 9.

An n to (m+t) CULM in the binary case.

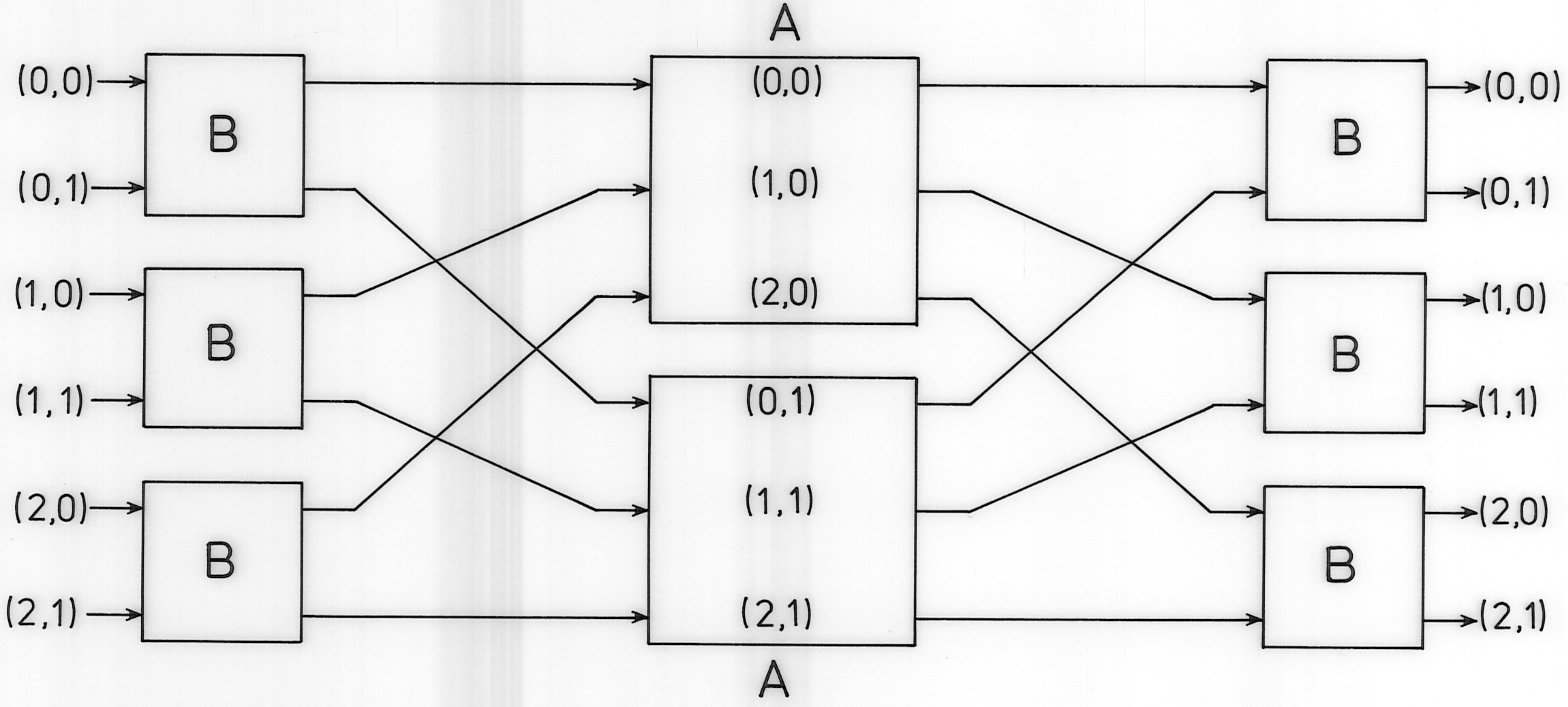


Figure 11. The Clos Network B××A

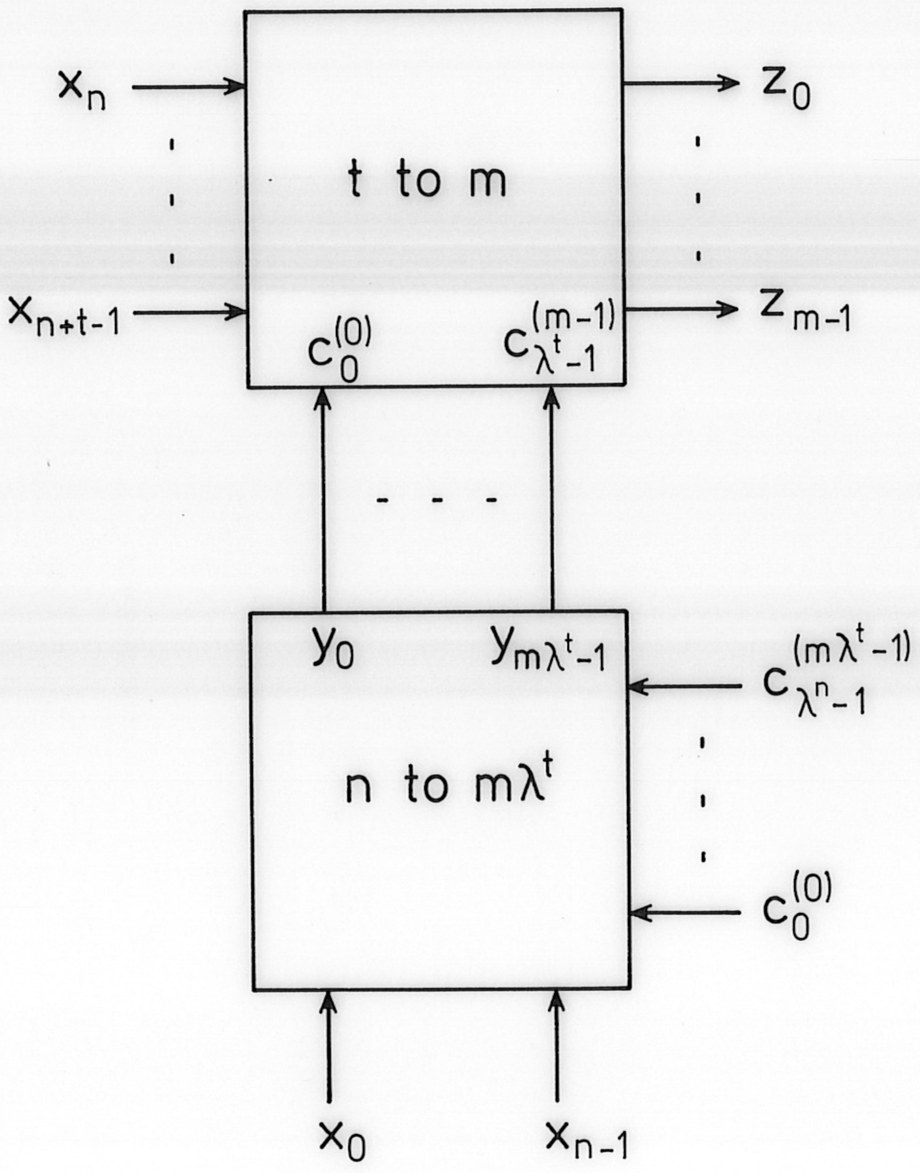


Figure 10.

An (n+t) to m CULM.

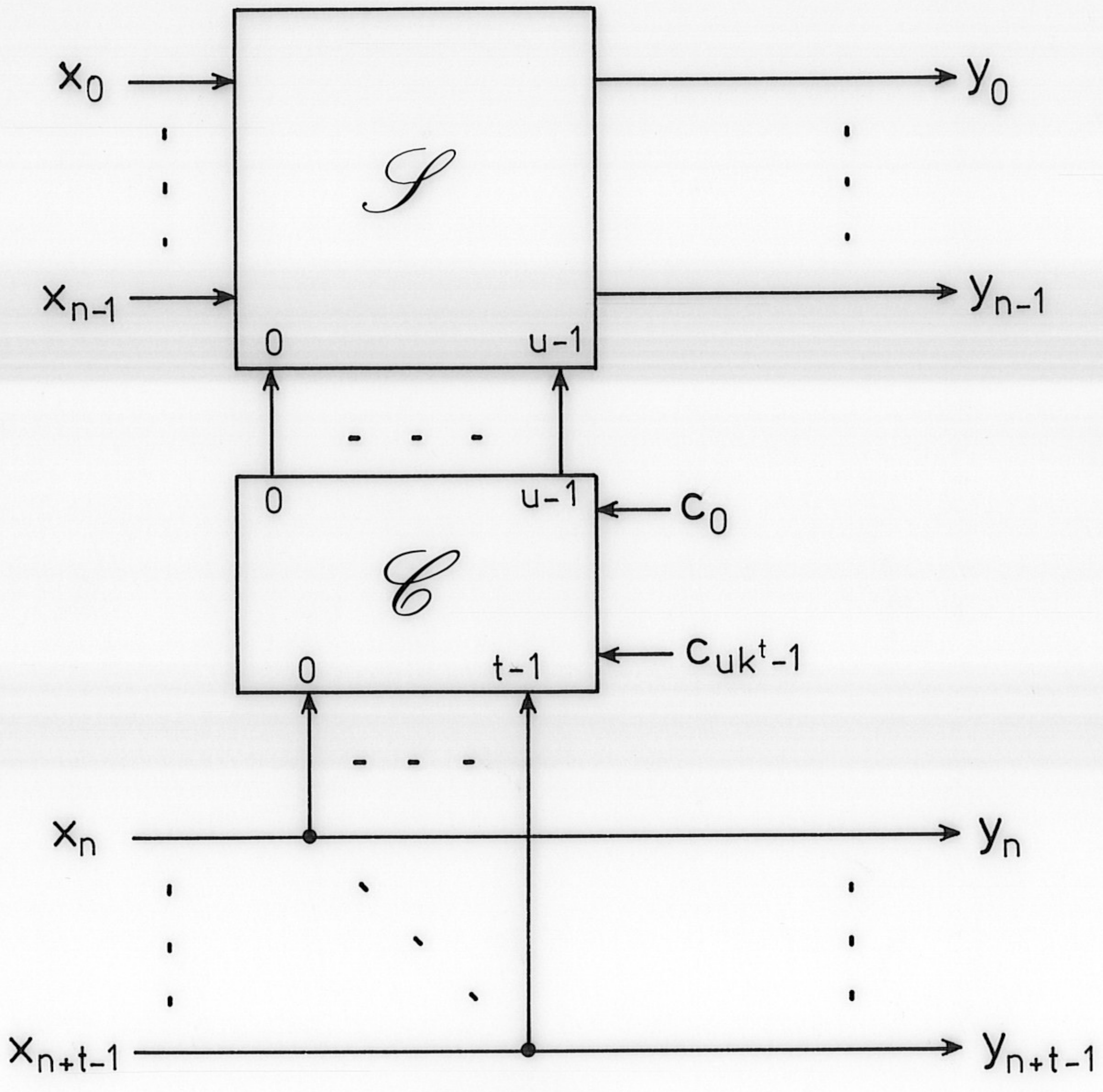
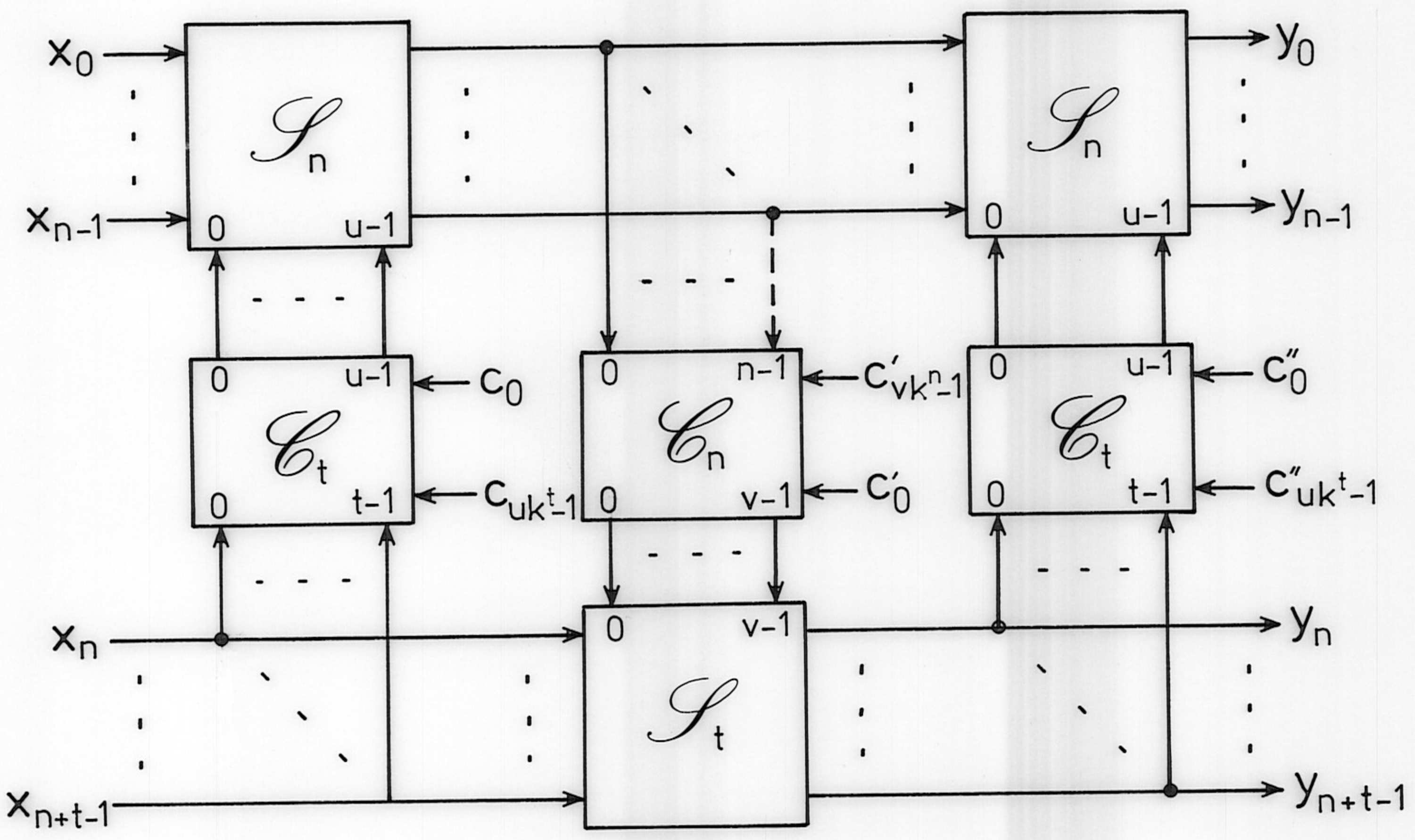


Figure 12.

$(u \geqslant |\log_k((k^n)!)|)$



$u \geqslant \lceil \log_k((k^n)!) \rceil \qquad v \geqslant \lceil \log_k((k^t)!) \rceil$

Figure 13. The three stage design of a total substitution network.

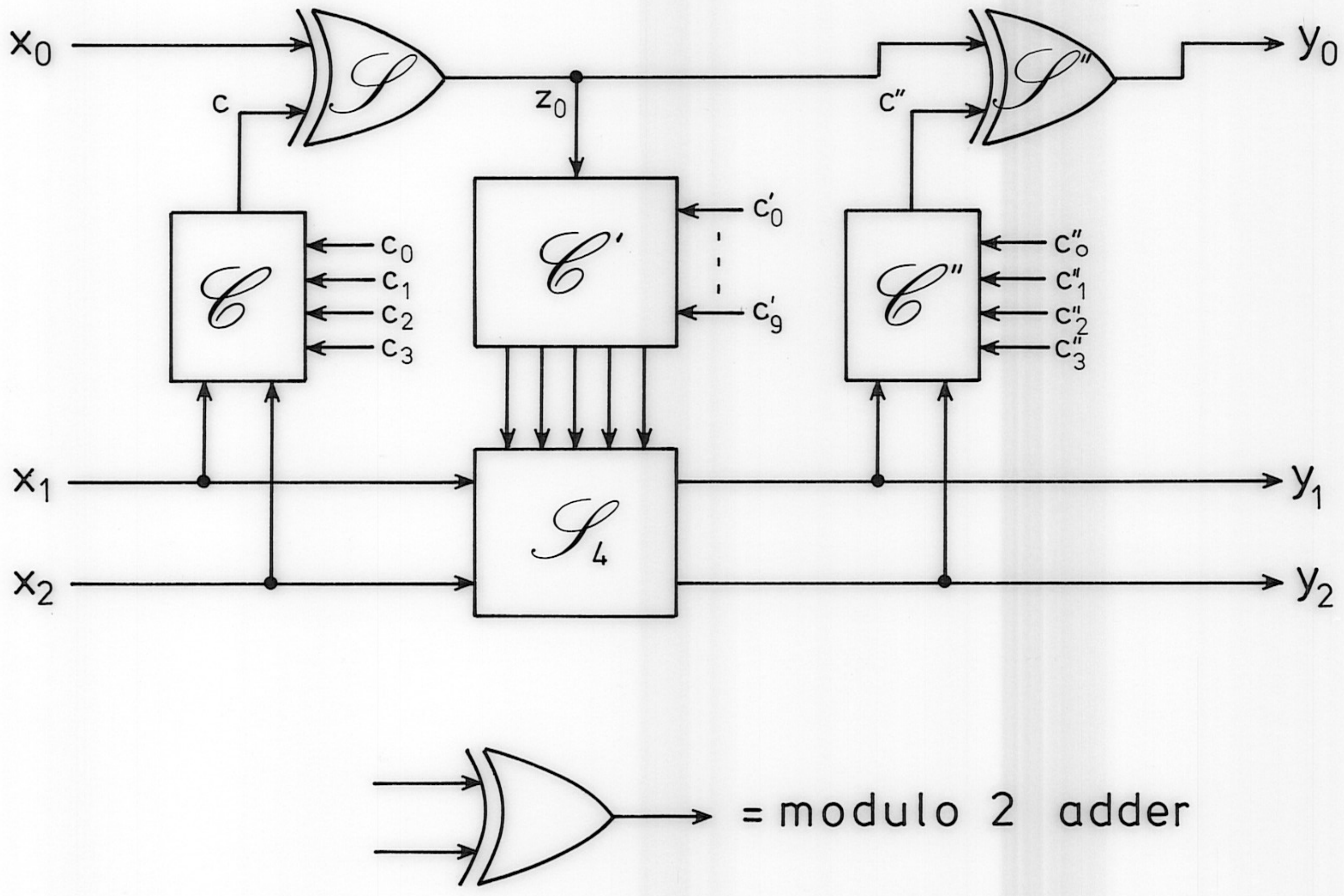

x_0
c
$\mathcal{S}$
z_0
$\mathcal{S}''$
c''
y_0
$\mathcal{C}$
c_0
c_1
c_2
c_3
$\mathcal{C}'$
c'_0
c'_9
$\mathcal{C}''$
c''_0
c''_1
c''_2
c''_3
x_1
x_2
$\mathcal{S}_4$
y_1
y_2
= modulo 2 adder

Figure 14.

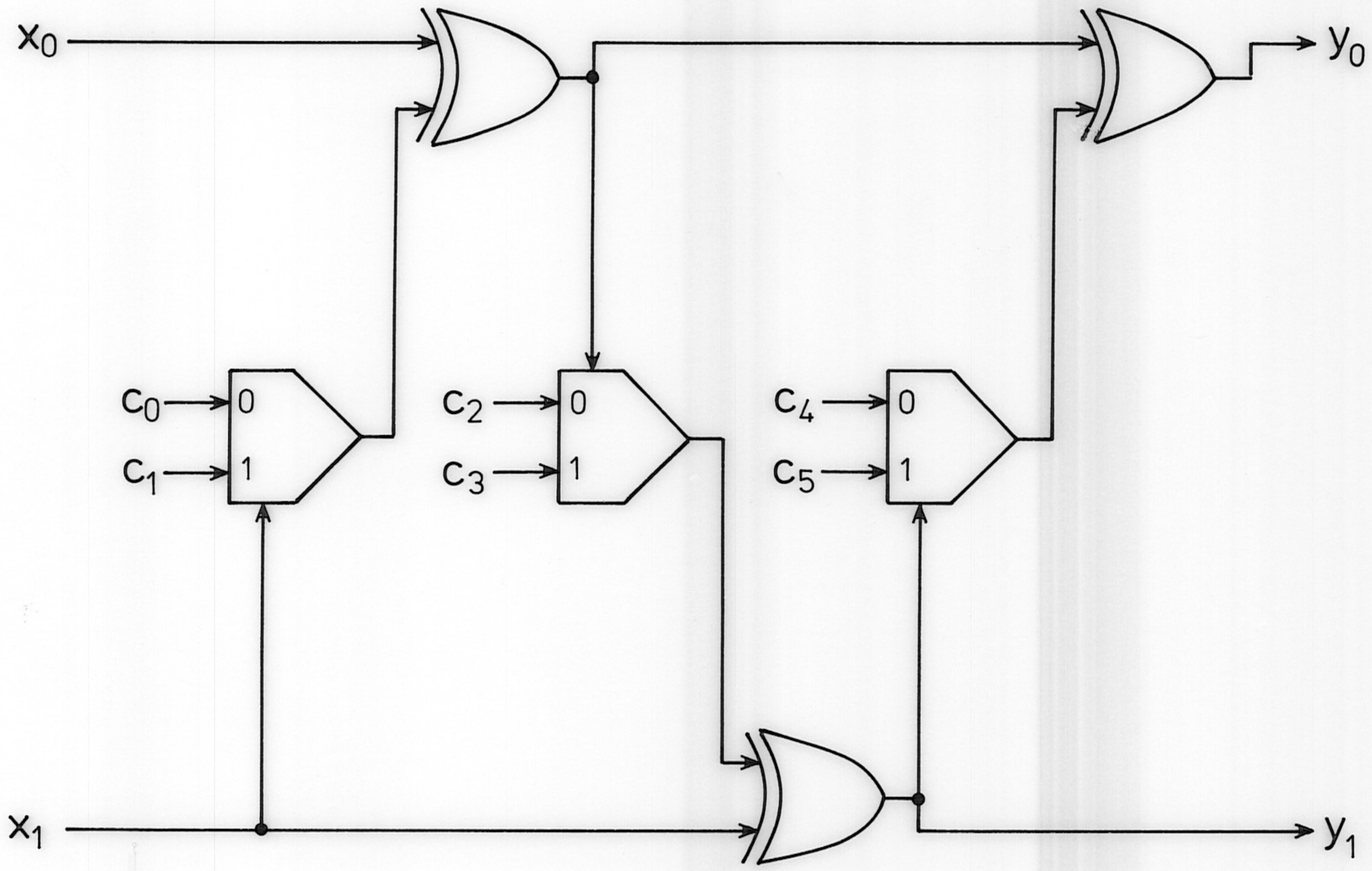
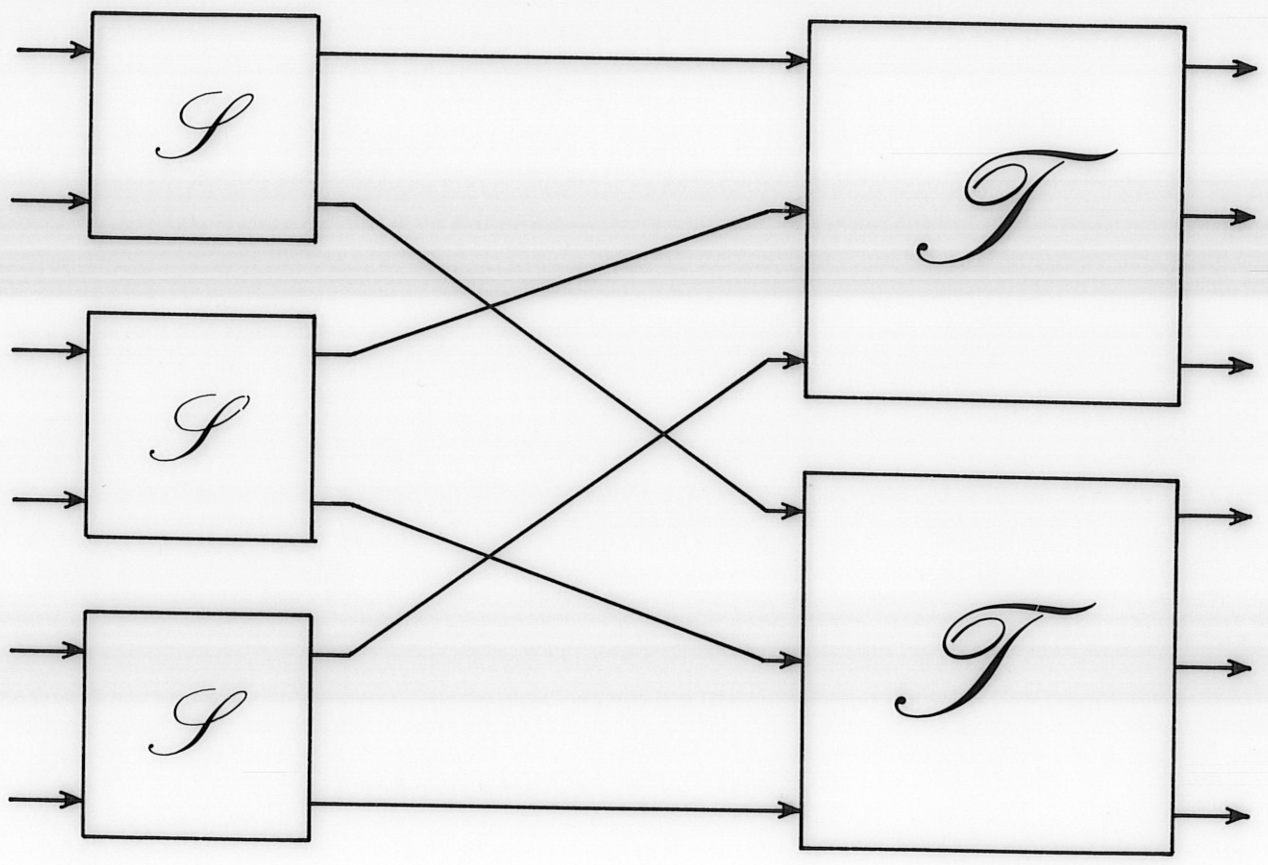
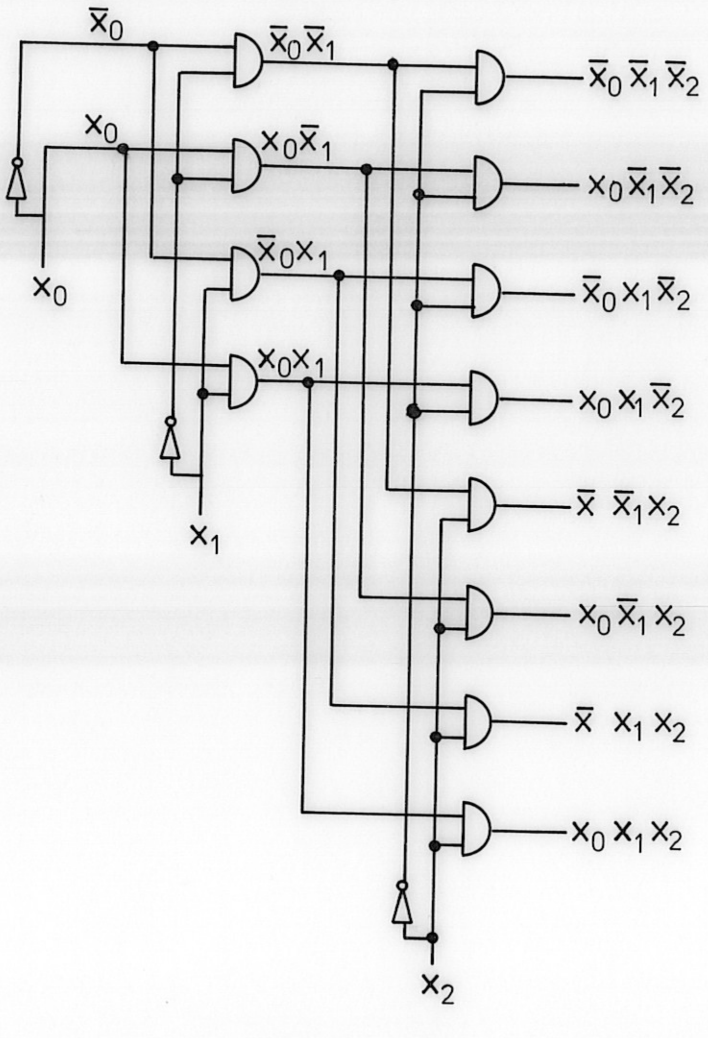


Figure 15. A total substitution network of size 2.



$\mathcal{S} \times \mathcal{T}$

Figure 16.



$\bar{x}_0$
x_0
x_0
$\bar{x}_0\bar{x}_1$
$x_0\bar{x}_1$
$\bar{x}_0x_1$
x_0x_1
x_1
x_2
$\bar{x}_0\bar{x}_1\bar{x}_2$
$x_0\bar{x}_1\bar{x}_2$
$\bar{x}_0x_1\bar{x}_2$
$x_0x_1\bar{x}_2$
$\bar{x}\ \bar{x}_1x_2$
$x_0\bar{x}_1x_2$
$\bar{x}\ x_1x_2$
$x_0x_1x_2$

Figure 17.

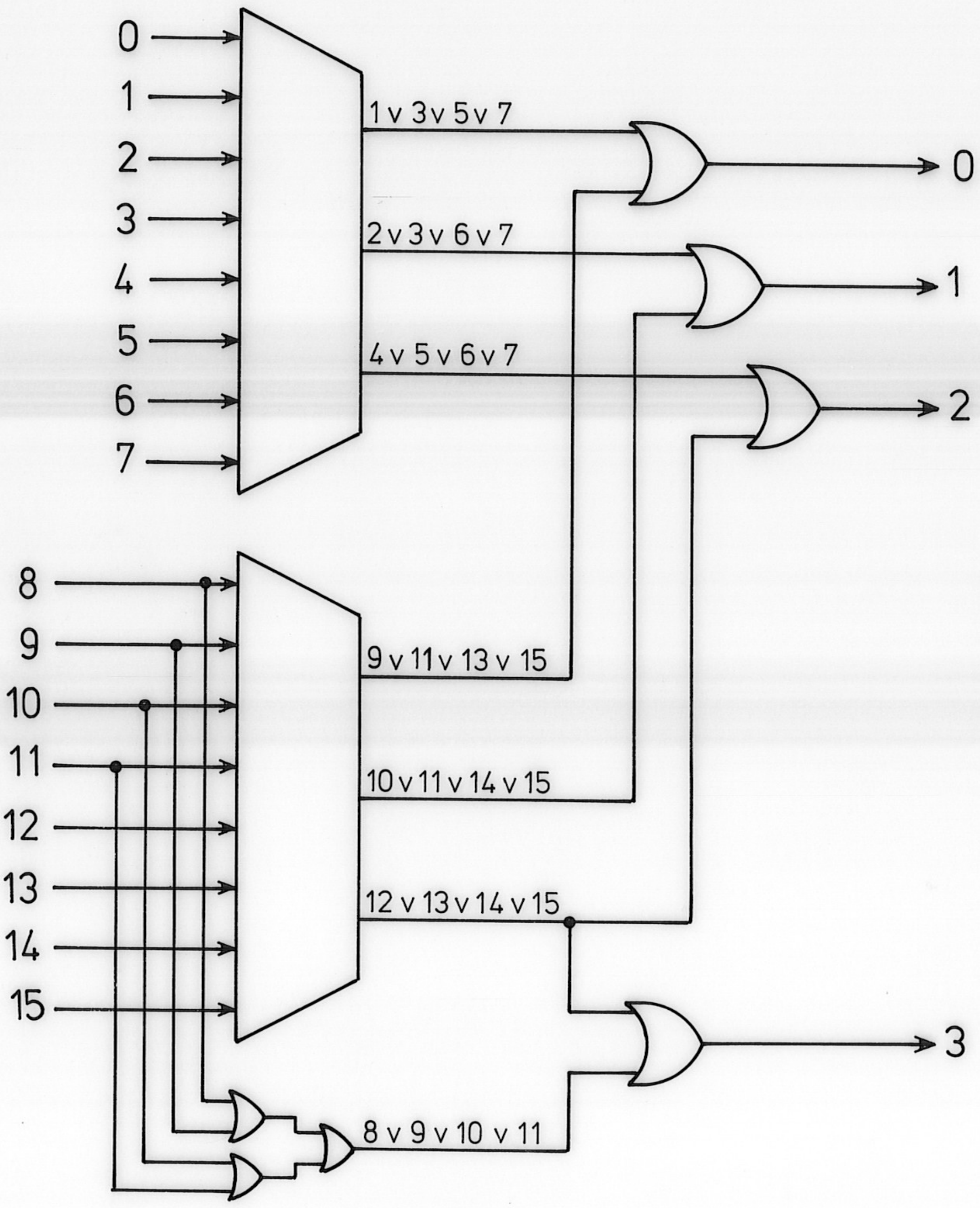


Figure 18.

A 16 to 4 address encoder

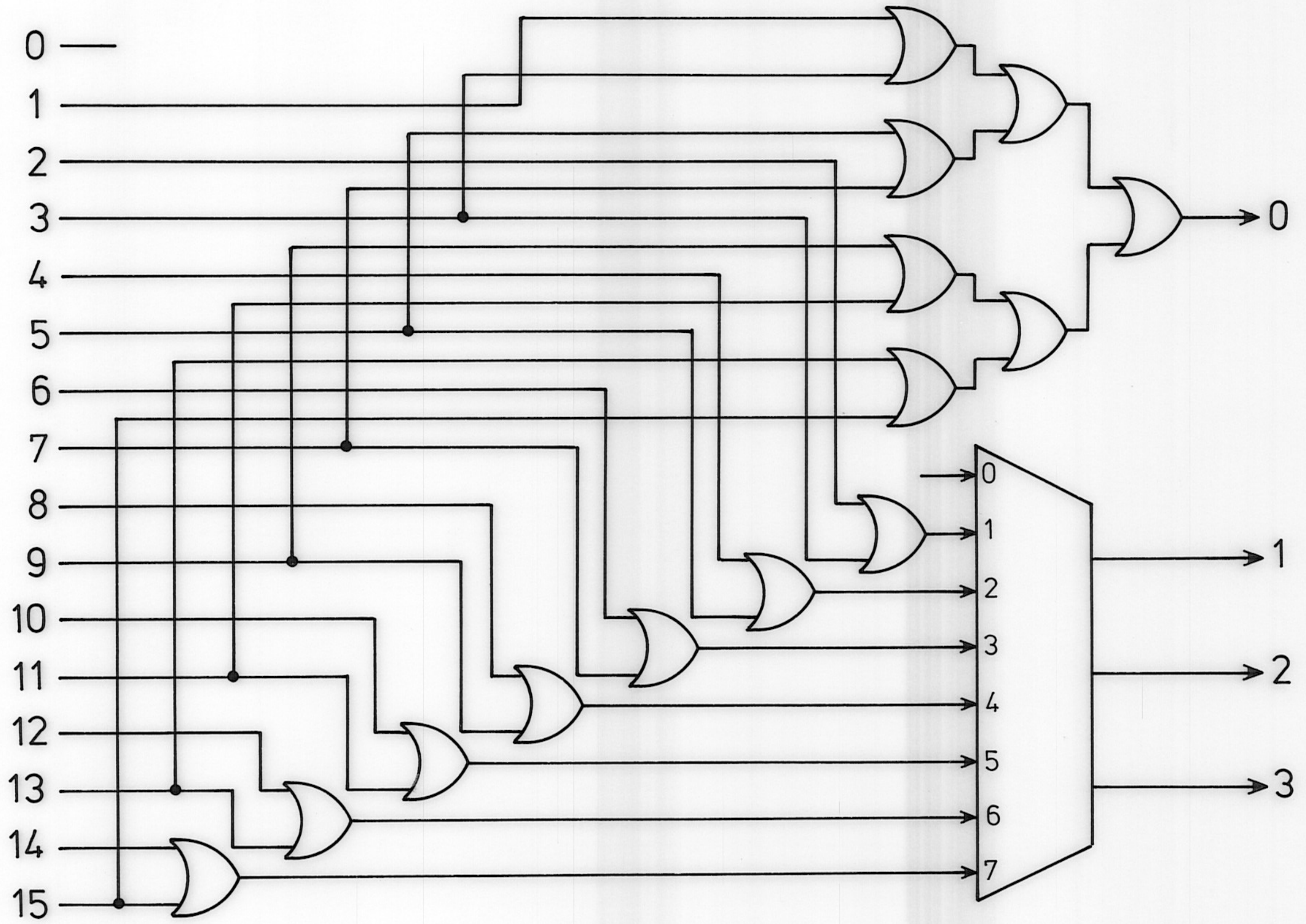


Figure 19. A 16 to 4 address encoder.

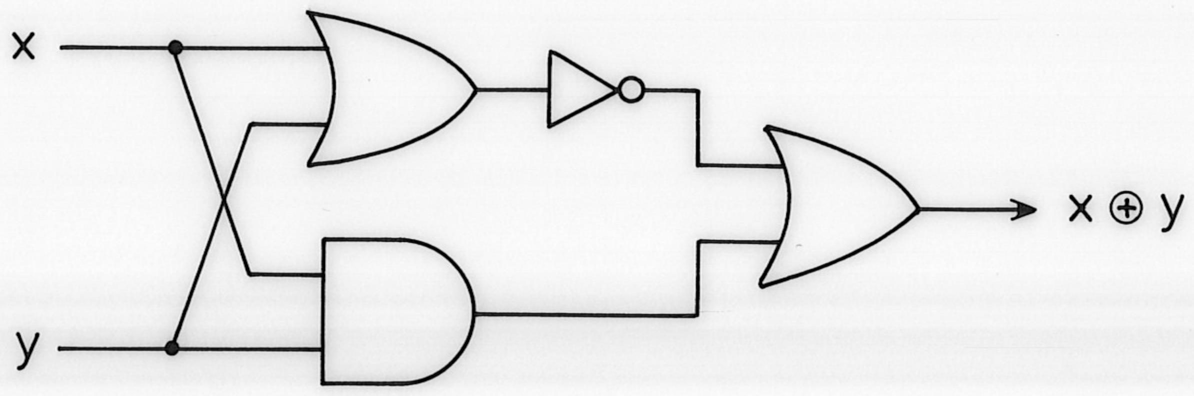


Figure 21.

A modulo 2 adder.

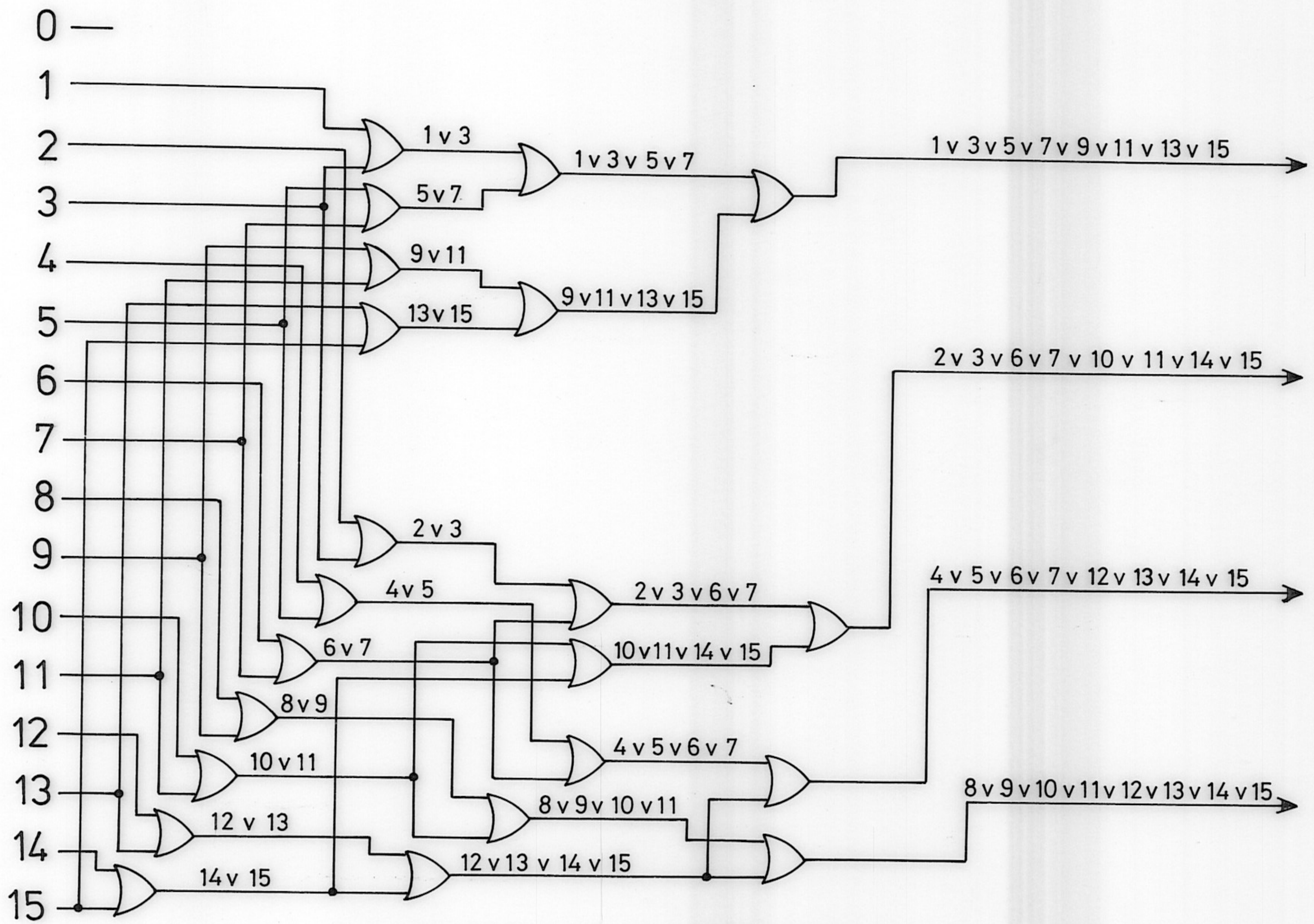


Figure 20. A 16 to 4 address encoder.

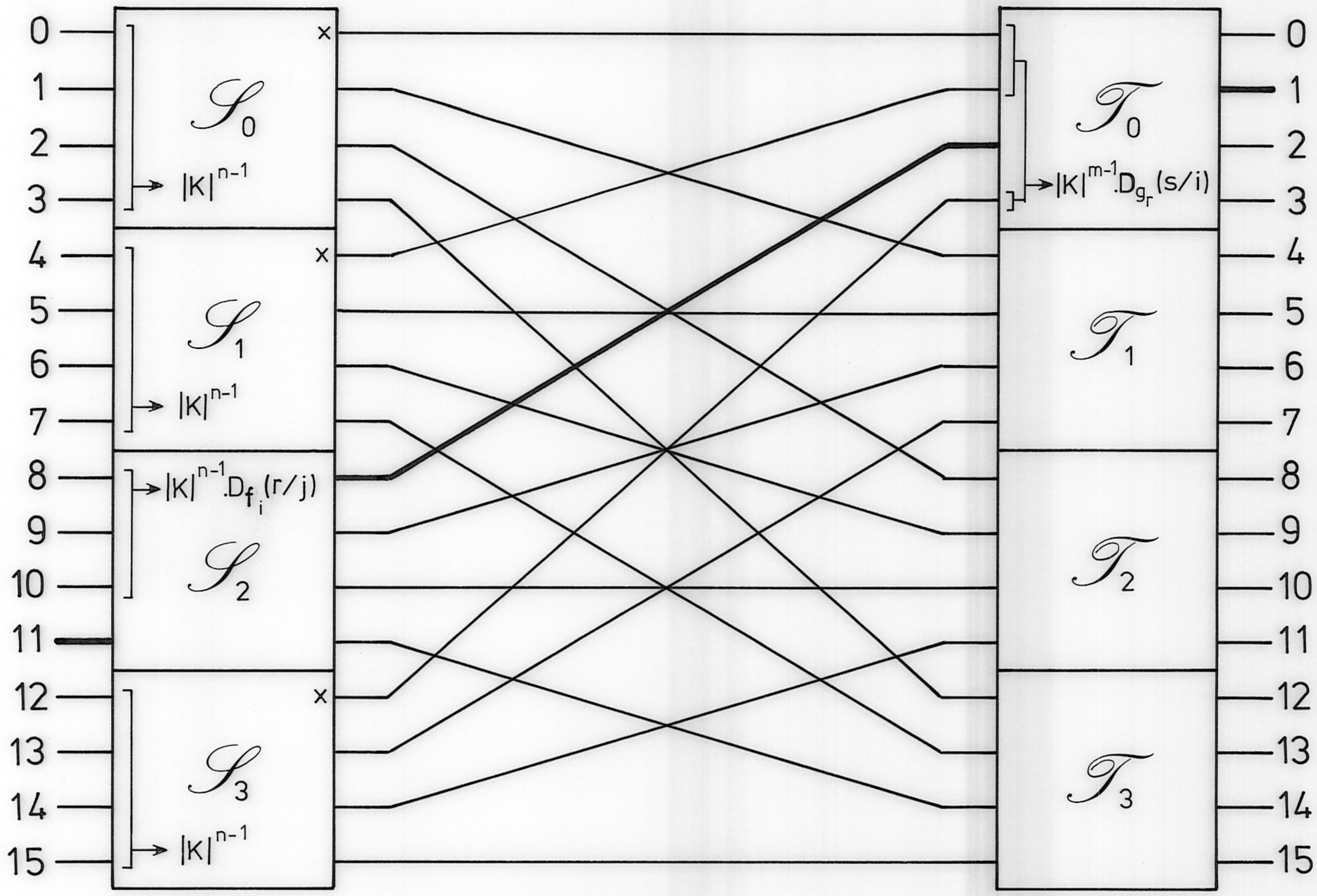

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
$\mathcal{S}_0$
$\mathcal{S}_1$
$\mathcal{S}_2$
$\mathcal{S}_3$
$|K|^{n-1}$
$|K|^{n-1}$
$|K|^{n-1}.D_{f_i}(r/j)$
$|K|^{n-1}$
×
×
×
$\mathcal{T}_0$
$\mathcal{T}_1$
$\mathcal{T}_2$
$\mathcal{T}_3$
$|K|^{m-1}.D_{g_r}(s/i)$

Figure 22.

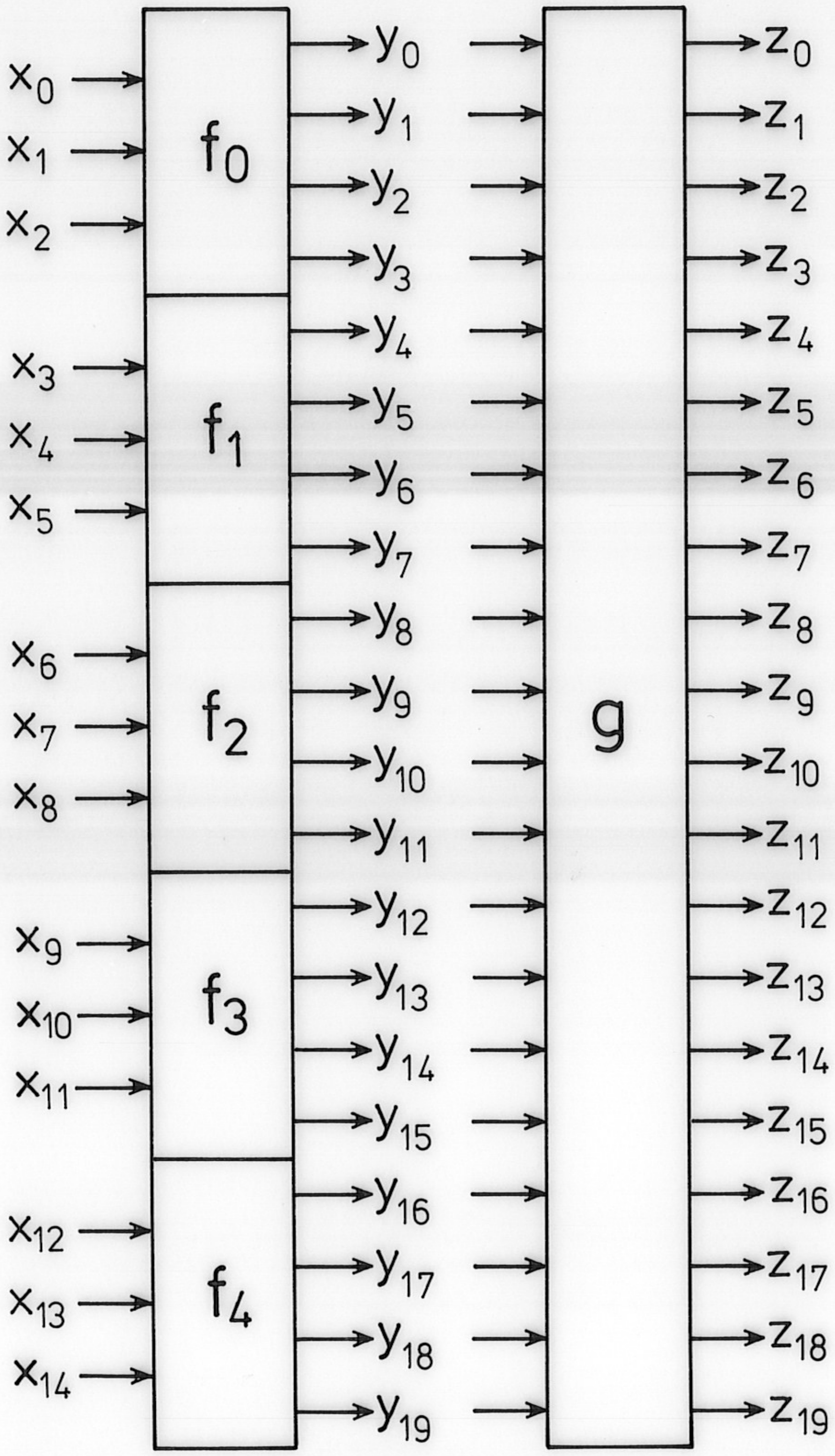
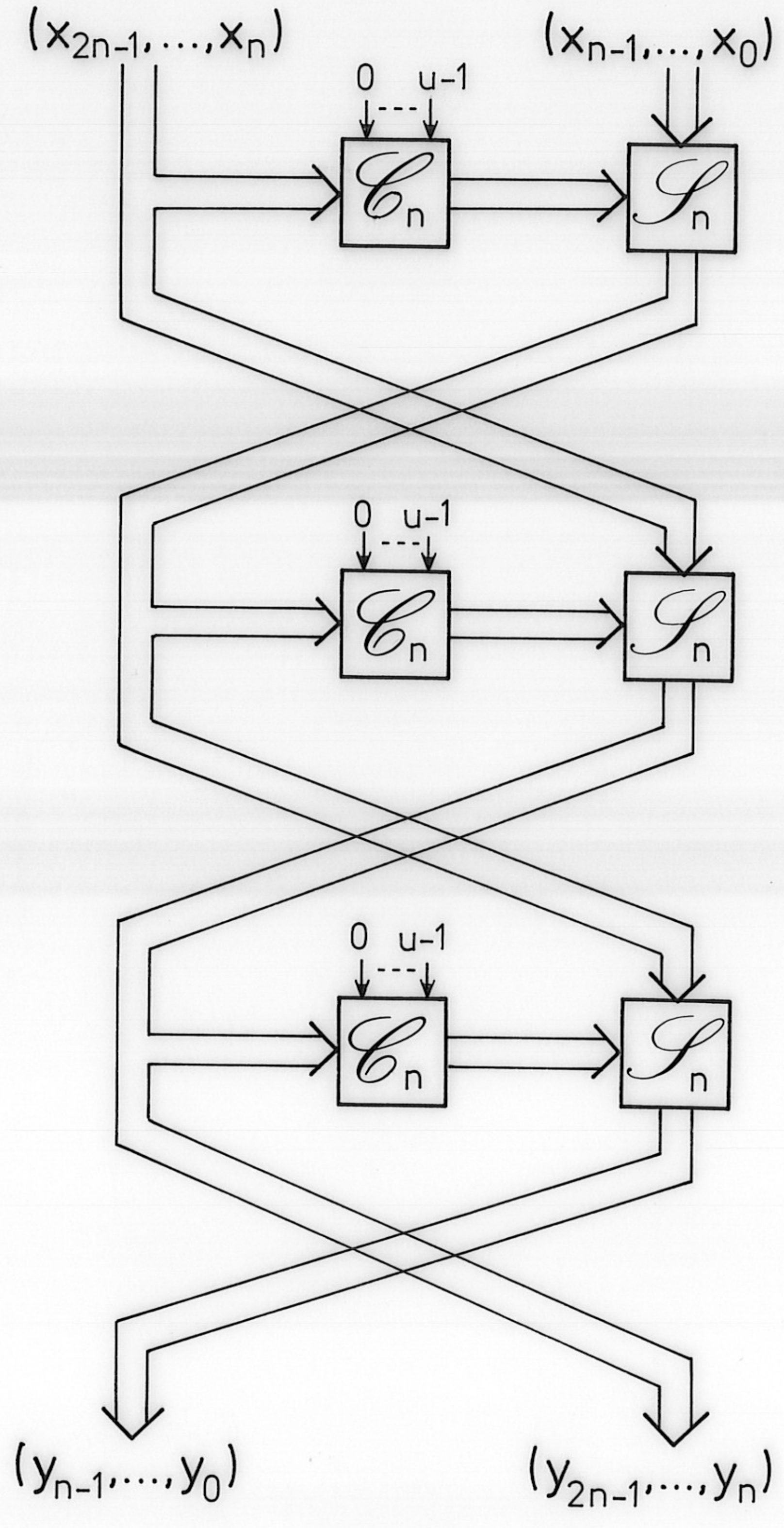


Figure 23.

$\lambda=3 \quad \mu=4 \quad n=5$



$(x_{2n-1},\dots,x_n)$
$(x_{n-1},\dots,x_0)$
0
u-1
$\mathscr{C}_n$
$\mathscr{S}_n$
0
u-1
$\mathscr{C}_n$
$\mathscr{S}_n$
0
u-1
$\mathscr{C}_n$
$\mathscr{S}_n$
$(y_{n-1},\dots,y_0)$
$(y_{2n-1},\dots,y_n)$

Figure 24.

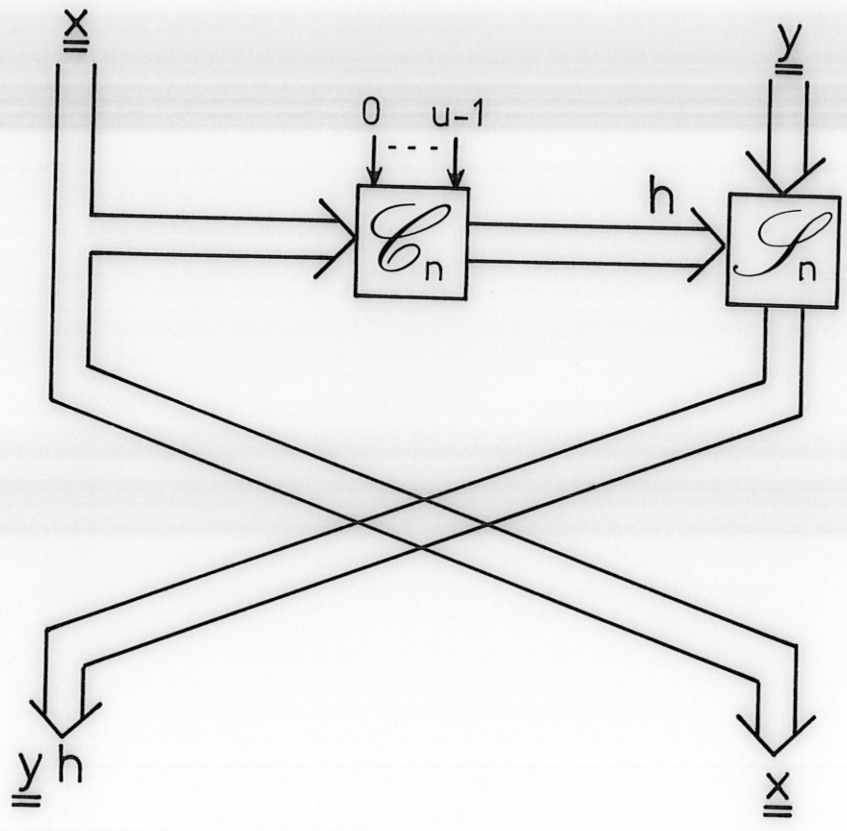

$\underline{\underline{x}}$
$\underline{\underline{y}}$
0
u-1
$\mathscr{C}_n$
h
$\mathscr{S}_n$
$\underline{\underline{y}}$ h
$\underline{\underline{x}}$

Figure 25.

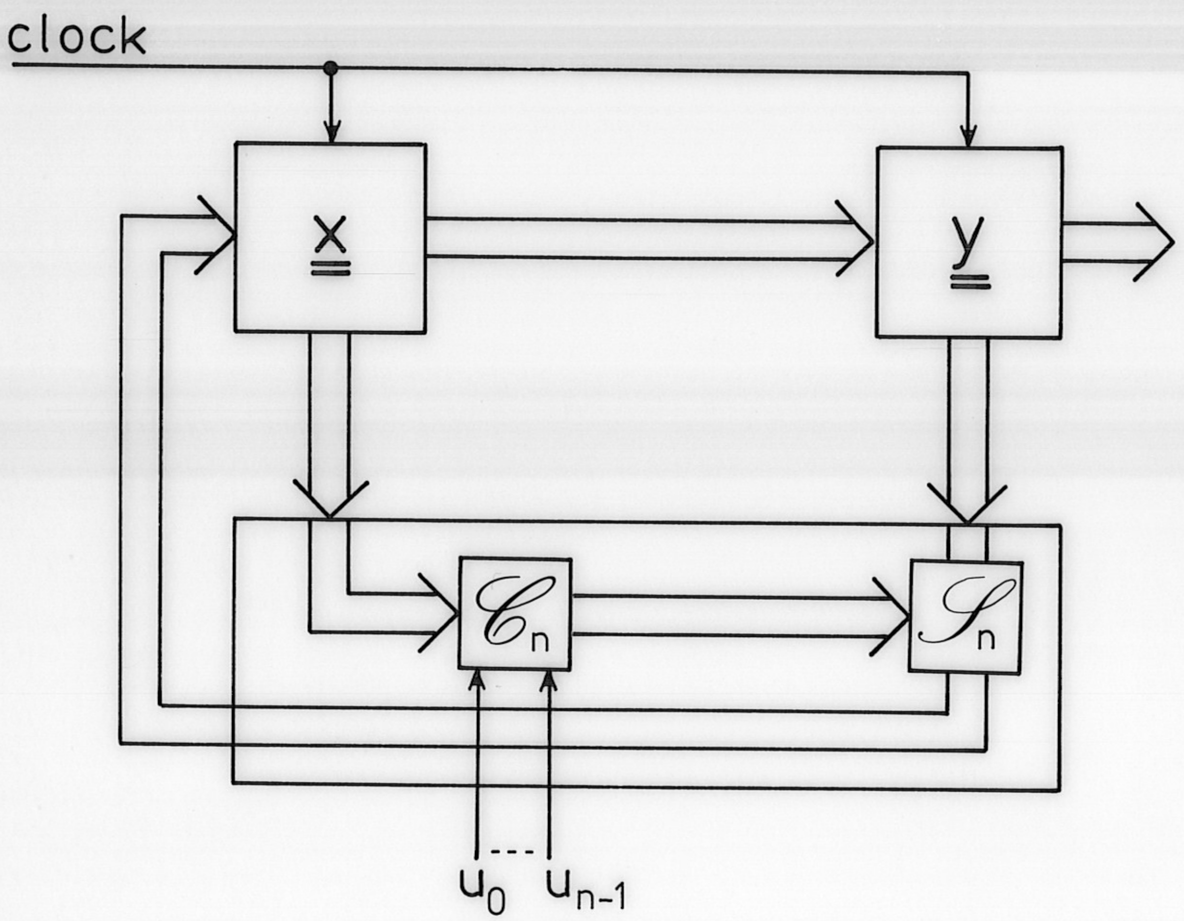

clock
$\underline{\underline{x}}$
$\underline{\underline{y}}$
$\mathscr{C}_n$
$\mathscr{S}_n$
u_0 --- u_{n-1}

Figure 26.