Poppelbaum, W.; Afuso, C.; Esch, J. (1967). "Stochastic computing elements and systems". Af i ps Fjc. 31: 635–644.

Stochastic computing elements and systems

by W. J. POPPELBAUM, C. AFUSO and J. W. ESCH
University of Illinois
Urbana, Illinois

INTRODUCTION

To date essentially only two fundamentally different representations of numbers have been used in electronic computers: One is the *analog representation* by a voltage or current inside a given range, the other one, the *digital representation* which maps a sequence of 0's and 1's onto a spatial or temporal sequence of voltage or current pulses. Of late, interest has arisen in the use of *random pulse sequences* as information carriers. (1), (2), (3), (4), (5), (6). It turns out that *the use of random pulse sequences leads to the use of digital ANDs or ORs for the fundamental operations of multiplication and summation* and therefore to a very considerable reduction in cost of the computational equipment. It is the purpose of this paper to present the theory of these random pulse sequences as well as their practical circuit implementations, and to give some systems design examples. In the final section some non-Von Neumann organizations will be discussed which are made possible by the use of stochastic computing elements.

In the course of developing the techniques in question our initial Random Pulse Sequence System (RPS) was replaced by the Synchronous Random Pulse Sequence System (SRPS). In the former case a sequence of standardized pulses represents the variable value by its average duty cycle, with the understanding that the standardized pulses occur at entirely random times. In the SRPS system these standardized pulses occur in fixed time slots with a probability of occurrence equal to the variable value. The main advantage of the latter system is that addition and multiplication of two SRPS's in an OR or an AND leads to another SRPS without any further normalization.

Figure 1 shows a way of generating controlled RPS's and SRPS's. The principle is simply to generate white noise in a noise diode and to amplify it in a threshold device, the threshold being proportional to the variable value. Reshaping of the output of this threshold differential amplifier leads to an appropriate RPS. For the production of an SRPS it is only necessary to set a flip-flop into the 0 state by the above RPS and to clear this flip-flop to 1 by a clock having a period considerably smaller than the average period of the RPS. The pulse transmitted through a capacitor upon resetting the flip-flop (if it had been set by the RPS during the preceding clock period) can now be reshaped in order to obtain an SRPS.

Figure 2 shows the inputs and outputs of an AND and an OR receiving as their input two SRPS's. It is easily seen that the average frequency of the outputs is respectively proportional to the sum or the product of the average input frequencies. It should be noted that certain difficulties (coincident pulses) arise in the case of the OR circuit when the incoming sequences are not mutually exclusive. This case can be easily eliminated by slightly more sophisticated design discussed in Section 4. A more detailed discussion of the mapping of numbers onto RPS's and SRPS's will be found in Section 2.

It is now clear that since adders and multipliers have the simple form of ORs and ANDs the production of very complicated linear combinations of numbers can be done with great ease and at low cost. The principal application of stochastic computing elements should therefore lie in those areas where such linear combinations are of use. Happily enough, the series expansion of the most general functions contain such linear combinations. One of the most attractive applications is that of a General Image Transformer described in Section 6.

It should be emphasized at this point that theory shows (See Section 2) that approximately 10,000 pulses are needed in order to obtain 1% precision. This means that a 1% result must wait for 1 millisecond if the average pulse repetition frequency is 10 MHz. This precision is entirely sufficient for most control applications and certainly more than adequate in the processing of light intensities, i.e., video information. Unluckily, a tenfold increase in precision, i.e., a

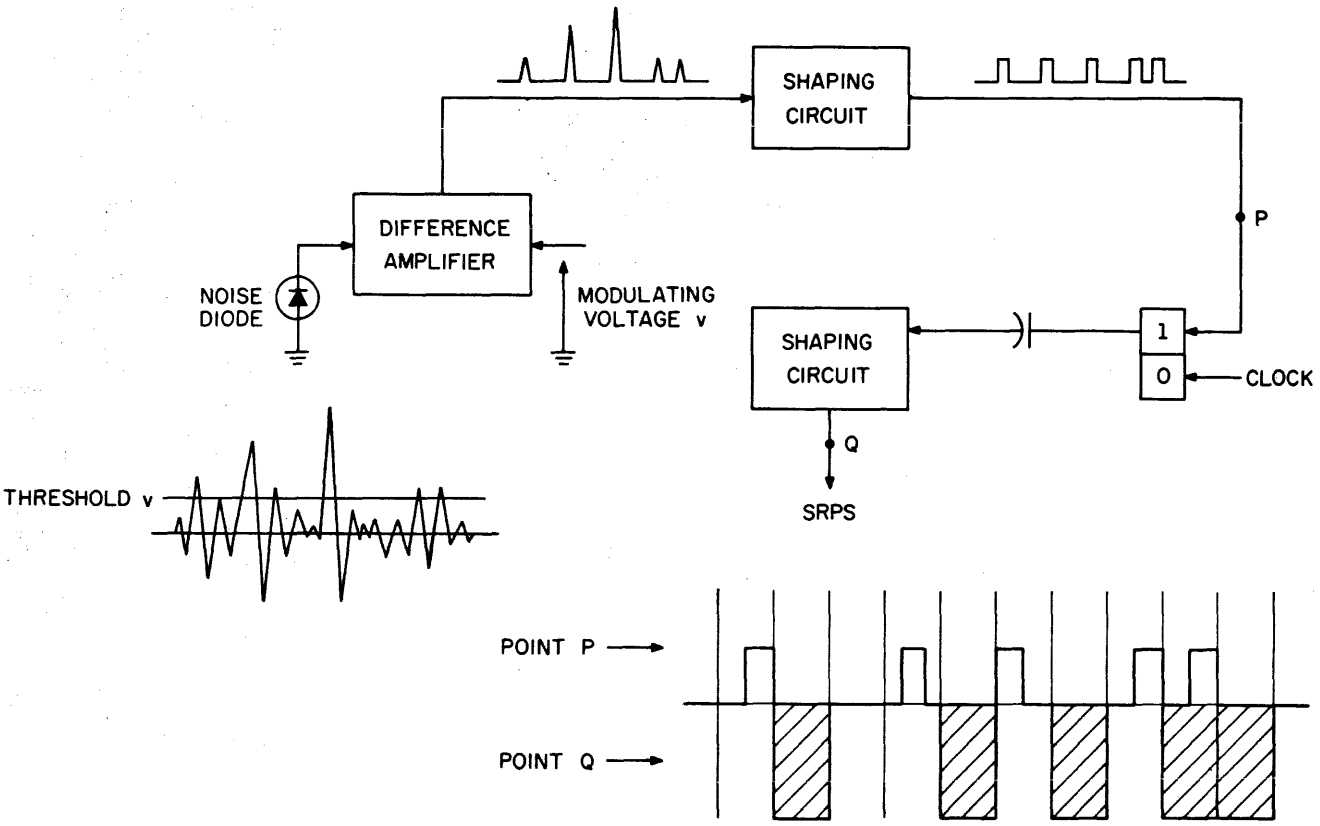


Figure 1 – Production of synchronous random pulse sequences

result known to within .1% necessitates not ten times more but 100 times more pulses. Stochastic computing elements are therefore strictly limited to calculations in which roughly .1% precision is the very upper limit. It should be emphasized that the limited precision in stochastic computing elements is inherent in the method because of the fluctuations encountered in the random process. This means that if we represent a constant value by an RPS or SRPS, the average frequency will vary slightly between adjacent time intervals. It is only when these time intervals are made long enough that the RMS value of the fluctuation becomes small enough. This still by no means excludes an entirely unlikely value for the average frequency in some very rare cases!

Mapping of numbers onto RPS's and SRPS's

In a RPS described by e(t) each pulse is of height V_0 and width τ and the frequency changes at random about an average value f: Figure 3 shows such a sequence. We can define the *average voltage V* over a time $T(T >> \tau)$ by

$$V = \frac{!}{T}\int_0^T e(t)dt \qquad (2.1)$$

and the average *duty cycle x* by

$$x = \frac{V}{V_0} \qquad (2.2)$$

Note that x is simply the probability at a given time of finding $e(t) = V_0$. It is easily seen that $0 \leq x \leq 1$ and that

$$f = \frac{x}{\tau} \qquad (2.3)$$

In the RPS-System it is natural to represent a normalized positive number X by $x = X$. Normalization is, of course, no restriction because any computer representation demands it. The restriction $X \geq 0$ can be lifted in two ways: We can invert the pulses so as to obtain negative values of V or we can use a separate wire to carry the sign information.

As we pointed out before, the shape of pulses is destroyed by passing them through OR's and AND's. This difficulty is eliminated by timing the beginning of each pulse, should it occur, by a centralized clock. The characteristics V_0 and τ of each pulse remain unchanged. Figure 4 shows such a SRPS and it is clear that AND- or OR-operation no longer affect the

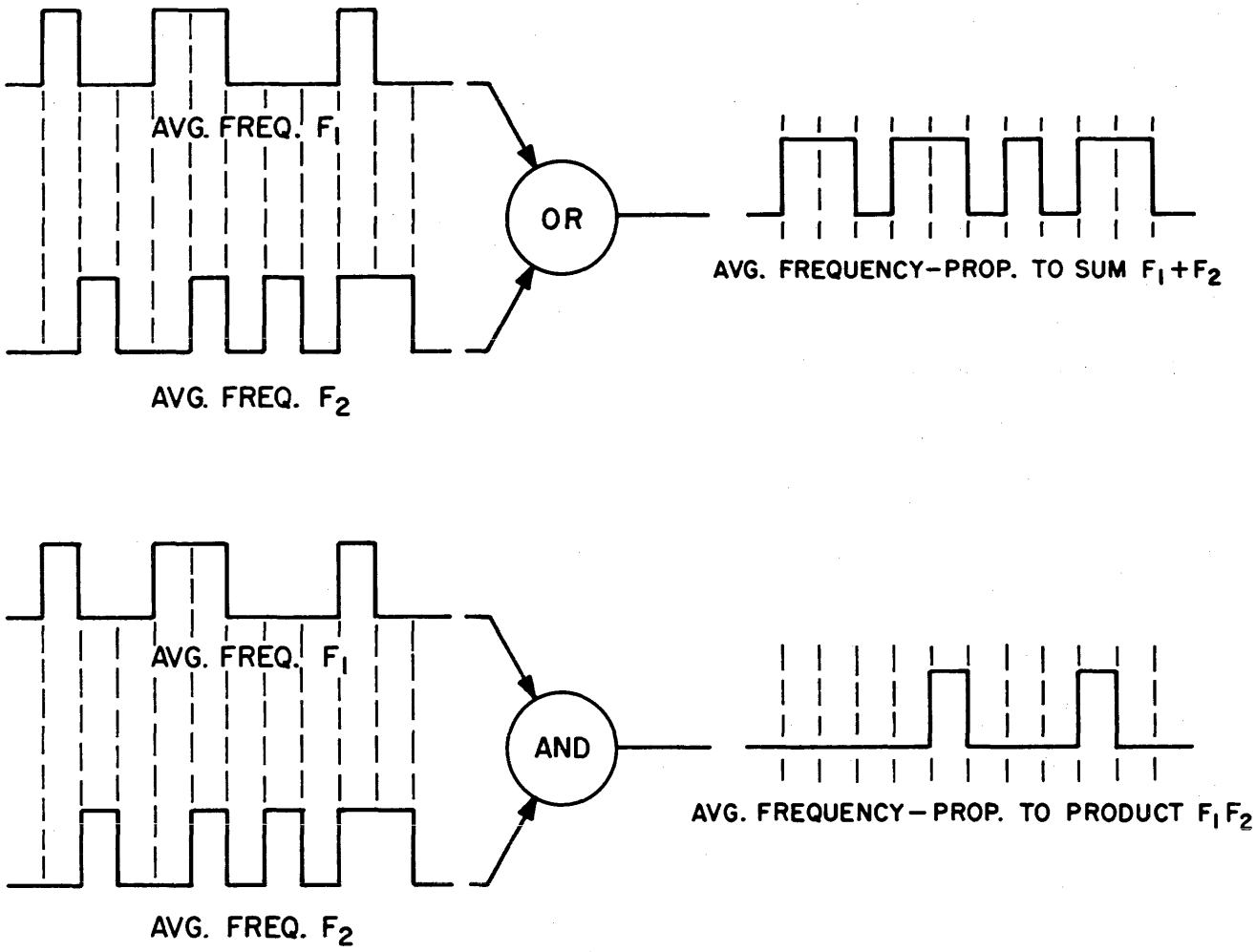


Figure 2 – Use of or's & and's for addition & multiplication in synchronous random pulse sequence system

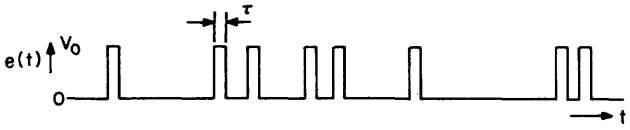


Figure 3 – A random pulse sequence

shape. It is also clear that equations (2.1)...(2.3) remain valid. Here, however, the maximum value of x is ≤ 1 because τ is usually less than the clock period T_o: It is more efficient to map X onto

$$u = \frac{f}{f_o} \text{ where } f_o = \frac{1}{T_o} + \tag{2.4}$$

i.e., the *probability* of the occurrence of a pulse in a given time-slot. All we have done, of course, is to replace $X \rightarrow x$ by $X \rightarrow u$ with

$$u = x\left(\frac{T_o}{\tau}\right) \tag{2.5}$$

Sign questions can again be solved as above.

In practice the averaging time T is finite and all quantities in (2.1)...(2.5) show fluctuations: These, as pointed out before, are inherent in randomness and limit the dynamic accuracy of the system. For a RPS with $\tau << T$ the pulse distribution in time is described by a Poisson distribution with standard deviation given by $\sqrt{fT}$. Therefore the relative fluctuation of all variable values with respect to the average value, fT, is

$$\Delta = \frac{100}{\sqrt{fT}}\,\% \tag{2.6}$$

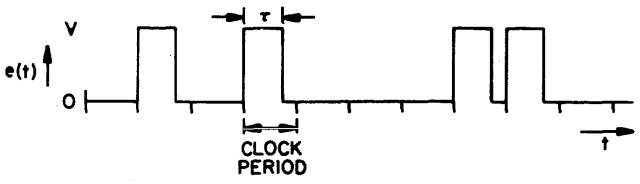


Figure 4 – A synchronous random pulse sequence

For a SRPS the probability distribution in the fixed time-slots is binomial with standard deviation given by $\sqrt{nu(1-u)}$. The relative fluctuation with respect to the average value, nu, is

$$\Delta = \frac{100}{\sqrt{n}} \cdot \sqrt{\frac{1}{u} - 1} \qquad \% \qquad (2.7)$$

where n is the number of clock pulses during T, i.e., $n = T/T_0$.

The four fundamental operations of arithmetic for RPS's

It is trivial to show that in the mapping $X \rightarrow x$ we can use OR's for addition *if we neglect the case of overlapping inputs*. The reason is, of course, that under this assumption we have simply $f = f_1 + f_2$ where f corresponds to the output and f_1 and f_2 to the inputs. This implies that

$$\frac{x}{\tau} = \frac{x_1}{\tau} + \frac{x_2}{\tau} \rightarrow x = x_1 + x_2 \qquad (3.1)$$

For multiplication we use the probabilistic interpretation of (2.2) which gives directly

$$x = x_1 \cdot x_2 \qquad (3.2)$$

for the output of an AND circuit fed by RPS's corresponding to x_1 and x_2 respectively.

Unluckily subtraction and division offer slightly more difficulty: They both necessitate the generation of an output RPS by a new RPS-generator, the latter being steered by an appropriate combination of average values. In the case of subtraction, we invert the sequence to be subtracted (i.e., form a sequence whose pulses go negative from ground) and use the difference of the average values of minuend and subtrahend to steer our generator. For division we use a layout as shown in Figure 5 in which the generator for x_3 is steered by a feedback system: When x_3 is such that x_2x_3 is on the average equal to x_1, we obviously have $x_3 = x_1/x_2$.

It is entirely possible to design a General Computing Element which performs any one of the above operations depending on a "function signal" from the outside world. Such a GCE is shown in Figures 6 and 7.

$G_1, G_2, \ldots, G_7$ are diode gates whose input impedance are practically infinite when they are OFF. For multiplication, G_1, G_3, and G_6 are ON (gate control voltage = 0) and the rest of the gates are OFF (gate control voltage = 5v if the pulse height of the R.P.S. is 5v). x_1 and x_2 are ANDed with D_1 and D_2 and the average voltage of the inversion of the product is compared with the average voltage of the inversion of the local R.P.S.G. R.P.S.G. is controlled by the comparator output such that the inputs to the comparator have the same average value.

For division, x_1/x_2, (assuming $x_1 < x_2$!) only G_2, G^3, and G_7 are ON. x_2x_3 is formed with D_2 and D_3 and the corresponding average voltage of x_2x_3 is compared with that of x_1 and the R.P.S.G. is adjusted so that $x_2x_3 = x_1$. The output, x_3, thus gives x_1/x_2.

For addition, only G_1, G_5, and G_6 are ON. Since G_3 and G_7 are OFF (5v) the AND gate formed with D_1, D_2, and D_3 gives x_1 itself. At the collector circuit of T_1 and T_7 the analog sum is formed and the R.P.S.G is adjusted so that $x_3 = x_1 + x_2$.

For subtraction, $x_1 - x_2$, assuming $x_1 \geqslant x_2$, only G_1, G_4, and G_6 are ON. The situation is the same as that of addition except that x_2 is fed through T_5 and T_6. Analog inversion takes place at T_6 and hence $x_3 = x_1 - x_2$.

Since R.P.S.G. employs 5 transistors and a few diodes, the whole unit employs 12 transistors and about 30 diodes. In order for the element to be able to handle negative inputs and outputs, additional sign carrying wires and sign detector circuits are needed.

The four fundamental operations of arithmetic for SRPS's

In the mapping $X \rightarrow x\left(\frac{T_0}{\tau}\right) = \frac{f}{f_0} = u$ we see without difficulty that we have again $f = f_1 + f_2$ for the output of an OR, i.e.,

$$u = u_1 + u_2 \qquad (4.1)$$

The probabilistic interpretation of u in (2.4) leads immediately to

$$u = u_1 \cdot u_2 \qquad (4.2)$$

for the output of an AND and therefore produces multiplication, this time without danger of damaging the pulse shapes.

Having in hand addition and multiplication by purely digital methods it becomes attractive to handle subtraction and division in a *purely digital fashion*. Happily enough this is possible in the case of SRPS's because of their occurrence in fixed time-slots. For subtraction we can build a simple *deletion circuit* in

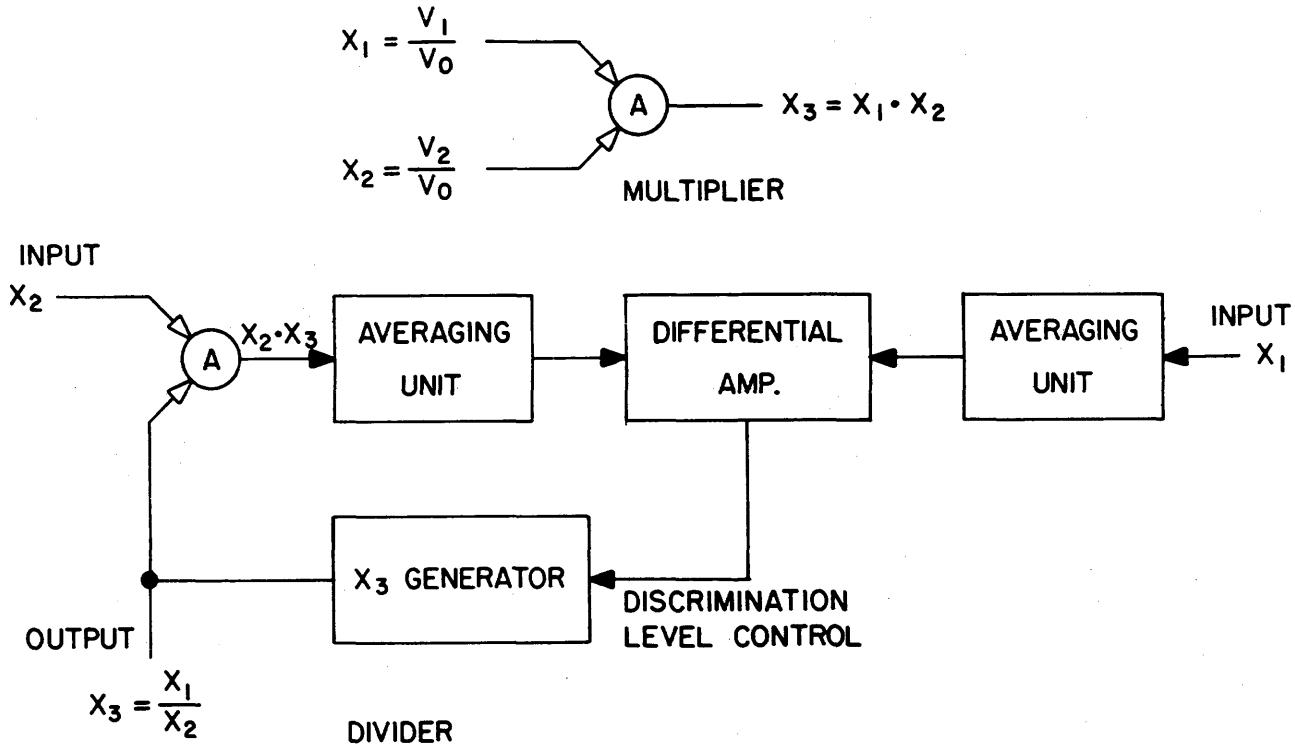


Figure 5 – Multiplier and divider in the R.P.S. system

which (for the case minuend > subtrahend) the occurrence of a subtrahend pulse leads to the deletion of the next minuend pulse. In order to obtain higher precision a small memory buffer may be incorporated so that bunching of subtrahend pulses can be taken care of. Note that the output of such a deletion circuit is again a SRPS and that obviously $f = f_1 - f_2$.

Division finally can also be handled digitally by observing that the problem X/Y can be solved for $X << 1$ and $0 \leq Y \leq 1$ by the following procedure:

1. Each time an X-pulse occurs, mark the next Y-pulse as the beginning of a "counting interval" and the Y-pulse following it as the end of a "counting interval."
2. Count all time-slots inside a "counting interval" and add 1.
3. The SRPS consisting of groups of bunched pulses in "counting intervals" represents $Z = X/Y$.

That this is effectively so, is seen by considering a very long sampling time T: in it there will be (See Figure 8)

$$n = Tf_0X \text{ X-pulses} \tag{4.3}$$

since $X = f/f_0$. The average period S of the Y-pulses is given by

$$S = \frac{1}{Yf_0} \tag{4.4}$$

This is by definition the length of a "counting interval" and corresponds therefore to Sf_0 pulses of the output sequence. During T we shall observe n bunches of Sf_0 pulses, i.e., the output represents a number Z such that

$$Z = \frac{\text{avg. frequency}}{f_0} = \frac{\text{\# of pulses}}{Tf_0}$$

$$= \frac{n(Sf_0)}{Tf_0} = \frac{(Tf_0X)}{(Yf_0)T} = \frac{X}{Y} \tag{4.5}$$

Again it is possible to design a GCE which, however, this time uses digital circuitry only. Figures 9-12 show a possible solution in which only the important paths are shown in a GCE consisting of

1. An UP-DOWN Counter.
2. A J-K Flipflop.

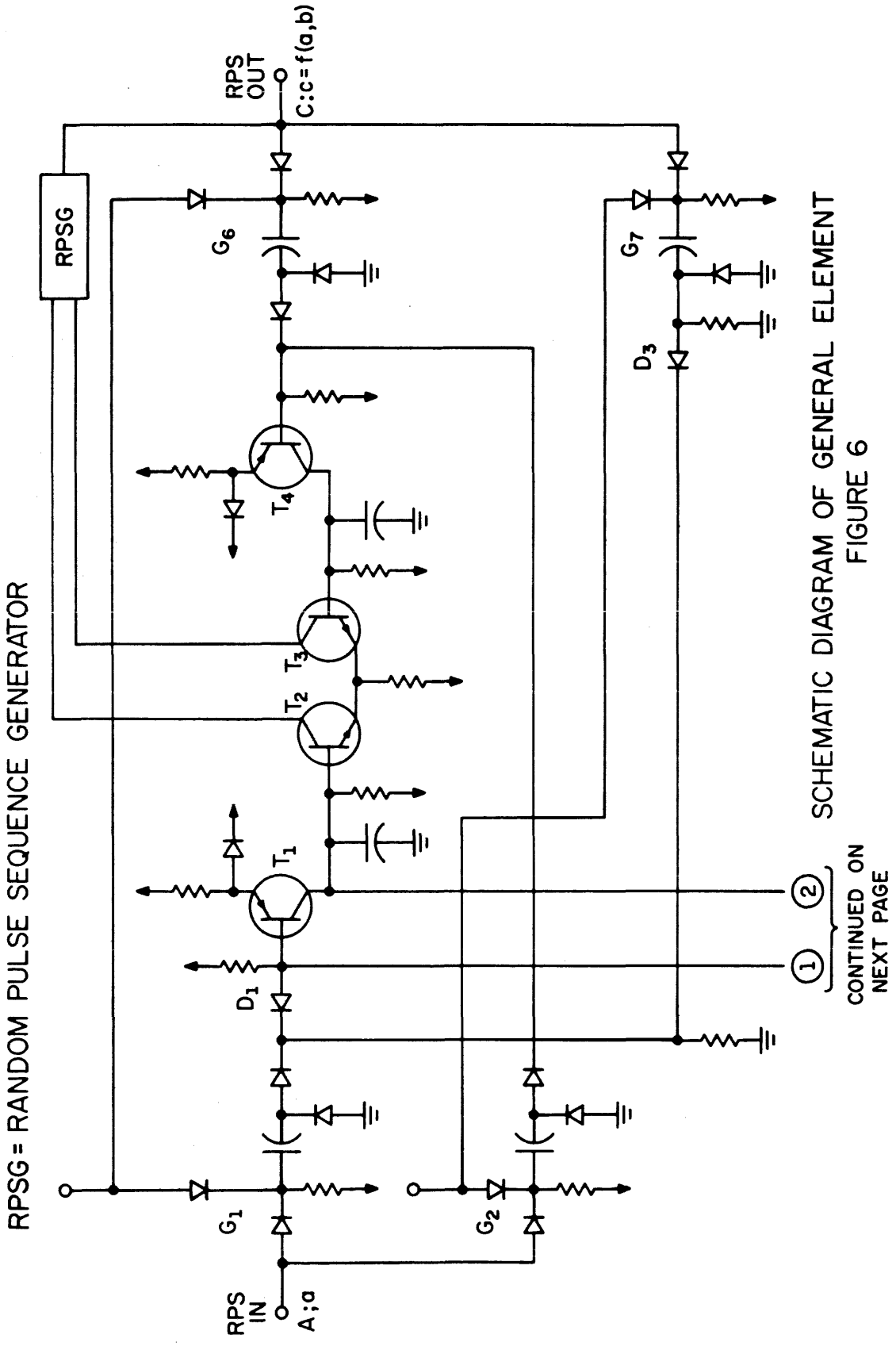


Figure 6 – Schematic diagram of general element

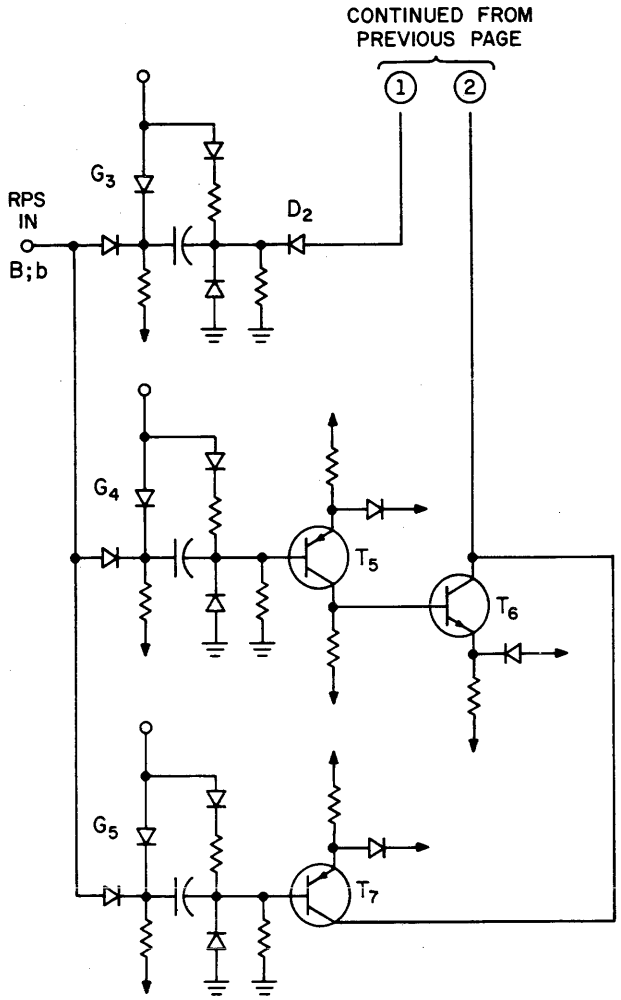


Figure 7 – Schematic diagram of general element

3. Digital Steering Circuitry.
4. Feedback Paths W and Z.

The system is visibly synchronous: The element puts out the sequence delayed by one clock period. In case of the division circuit an additional refinement injects a normalizing factor .1 so that we can use $0 \leq X \leq 1$ and $0 \leq Y \leq 1$ with the single restriction of $X \leq Y$.

Again operation with negative numbers is possible but will not be discussed at this point.

An array computer using GCE's

By a slight extension of what was discussed in the previous section, we can define a computing element which performs on its two inputs a and b the operations a + b, a − b, a × b, as well as two types of division a÷ b (denoted by /) and b ÷ a (denoted by /). To this repertory we shall add the operation (A) → connection of output to a and (B) → connection of output to b.

It is then possible to design array computers of considerable power having a fixed structure. Figure 13 shows a three row, eleven column array with a "straight through" connection for the top row and a "down right" connection for the bottom rows. Programming this array now simply consists in defining the function of each GCE for the operation. An interesting example is furnished by functions F which can be expressed as sums of continuing fractions, i.e., $F = \Sigma\ f_i$ where f_i = continuing fraction. In this case there exists a simple 5-step algorithm to determine the operation to be performed by each element:

Step 1.: Using a "Polish Notation" in which operands precede operators and using the symbols + ,−, /, /, (A) and (B) defined above, write Σf_i as a "Polish String",PS, with (B) inserted after the first f_i.

Step 2.: Write each f_i as a PS with all products treated as single variables (called "product variables") and insert (B) after the first variable or "product variable" of the f_i.

Step 3.: Write each "product variable" of Step 2 in terms of individual variables in a PS. Insert (B) after each variable which is not a "product variable" and each first variable of a "product variable."

Step 4.: Combine Steps 1 through 3 into a single PS.

Step 5.: "Fill in" the functions of the elements by reading off the PS of Step 4. To this end start with the bottom left-hand element of the array and procede horizontally. When more than one operator occurs consecutively, fill in vertically until the next operand is reached. At the end, fill in all unspecified elements with the operator (A).

The figure shows as an example the set-up for the function

$$F = r/(s + t) + (gh)/\{(wt) + b\ [(dp) + y]\}$$

It is easily verified that the algorithm leads to the PS given below the expression for F and that the array effectively forms F.

A stochastic image transformer

As mentioned in the Introduction, one of the most attractive applications of stochastic computing elements is the simultaneous formation of very extensive linear combinations of variable values. We shall now describe a machine organization (which is as far removed from a Von Neumann machine as possible) which uses exclusively such linear combinations. Although we call this system an Image Transformer, it should be realized that "Signal Transformer" would be more appropriate. Figure 14 shows the schematic diagram of what is proposed: The object of the game is to map n^2 signals x_{ij} $(i,j = 1, \ldots, n)$ onto n^2 output signals y_{kl} $(k, l = 1, \ldots, n)$ in such a fashion that each one of the y's is a weighted linear combination

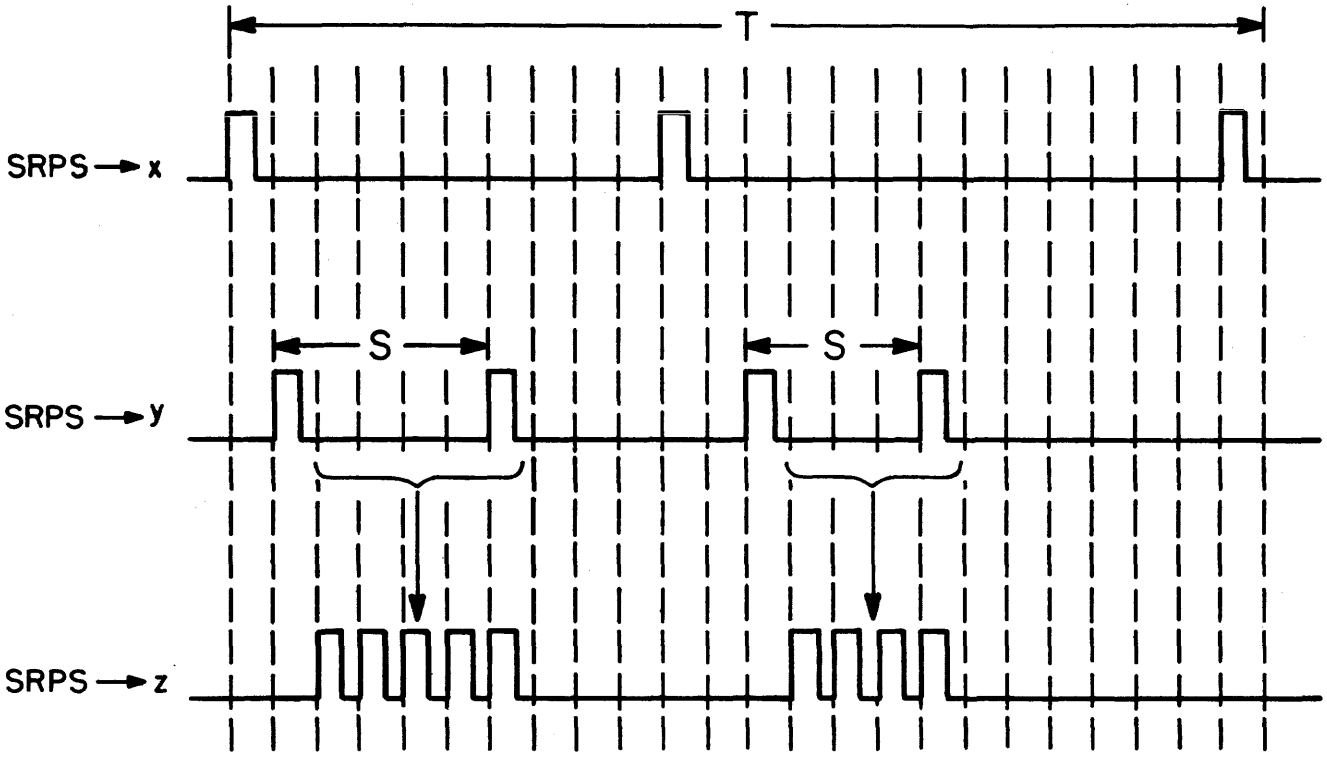
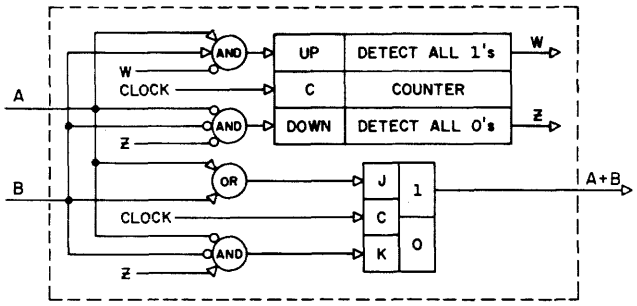


Figure 8 – Division of SRPS's



Fgure 9 – Addition of SRP's A and B

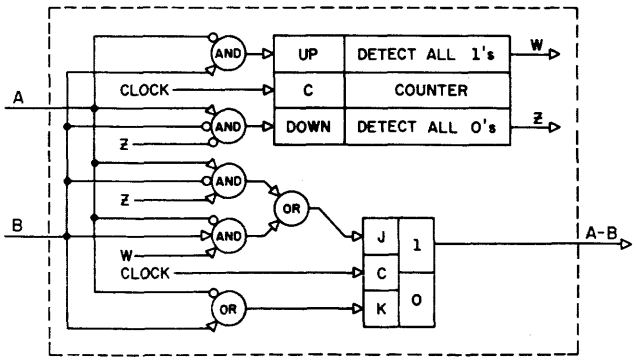


Figure 10 – Subtraction of SRPS B from SRPS A

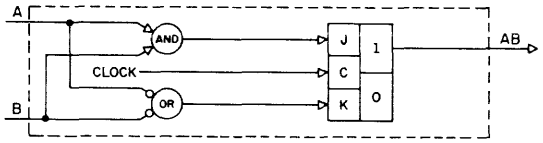


Figure 11 – Multiplication of SRPS's A and B

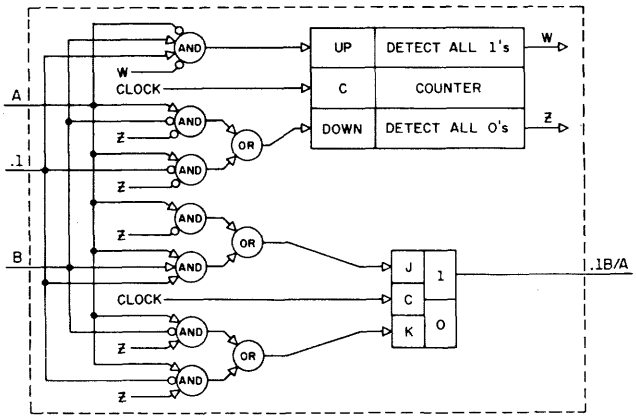


Figure 12 – Division of SRPS B by SRPS A and scaling

F = r/(s+t)+gh/(tw+b/(dp+y))

F = t(B)(B)s(B)+r(B)\(B)y(B)(B)p(B)dx+b(B)\t(B)wx+h(B)gx\+

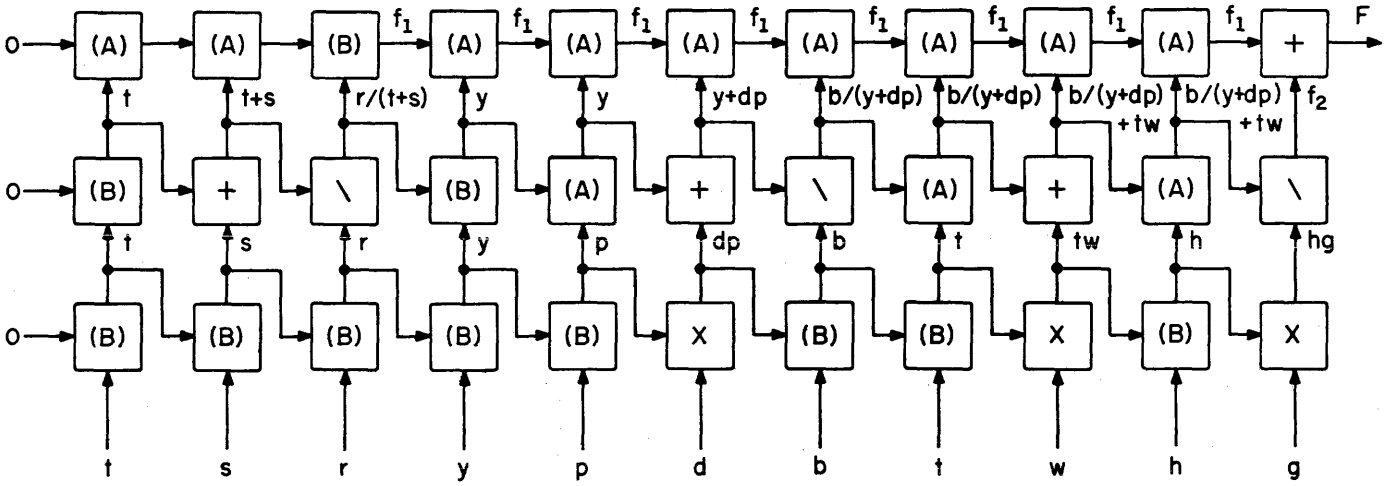


Figure 13 – Array of stochastic computing elements

of all x's. This corresponds to $y_{kl} = \sum_{ij} b_{klij} x_{ij}$. Evidently this mapping is not the most general possible but can be considered – after addition of a constant term a_{kl} – as the linear portion of the expansion of the most general function $f_{kl}(\ldots, x_{ij}, \ldots)$. It is then clear that the system produces an image mapping from input to output described by a general linear relationship and defined by the n^4 co-efficients b_{klij}. This general transformation encompasses:

1. Translations, rotations, and magnifications.
2. Conformal mappings.
3. Convolutions, and
4. Fourier Transforms.

By inspection it is easily seen that for a parallel organization in which each one of the n^2 output points collects its information through n^2 2-input AND's (each AND having x_{ij} and b_{klij} as inputs where k, l corresponds to the output point chosen), n^4 AND circuits will be necessary. Note that with respect to these the n^2 OR circuits can be neglected. The minimum number for n in a practical example would be 32: Even then over a million AND circuits would be required. This shows that a parallel organization is probably too costly. A way out is to consider a serial organization in which each of the output points is connected sequentially to n^2 AND circuits as described above. In this case higher circuit speeds are required since the on-line operation demands that all n^2 output points be scanned inside one flicker period, i.e., a 30th of a second. With circuit speeds of the order of 30 megacycles and demanding 10% accuracy (i.e., integration over 100 pulses) it is easily seen that the maximum n for the output matrix is about 200. This is not too far from television definition and it seems possible to reach a compromise in which the sequential system would be comparable to on-line video processors in both definition and speed. It should be noted, however, that the sequential scanning of the output necessitates a decoding matrix involving another n^2 (3-input) ANDs as shown in Figure 15, the inputs being the driving pulses for the two coordinate axes and the RPS coming from the AND combination described above. Note that there is perfect compatibility between the matrix selection pulses on the k and l lines and the stochastic signal carrying the information. It is now clear that the system will only require $2n^2$ ANDs and that the above mentioned figure of N = 200 leads to less than 100,000 AND circuits, a number not incompatible with cost considerations. The integrator at the output can be a tungsten light.

It is interesting to note that the determination of the b_{klij}'s hinges on the realization that i and j are (quantized) input coordinates and k and l (quantized) output coordinates. Assuming i, j, k and l continuous, we would therefore write a general transformation as

$$\left.\begin{aligned} k &= k\ (i,J) \\ l &= l\ (i,j) \end{aligned}\right\} \qquad (6.1)$$

The identity transformation k = i and l = j gives, obviously, rise to

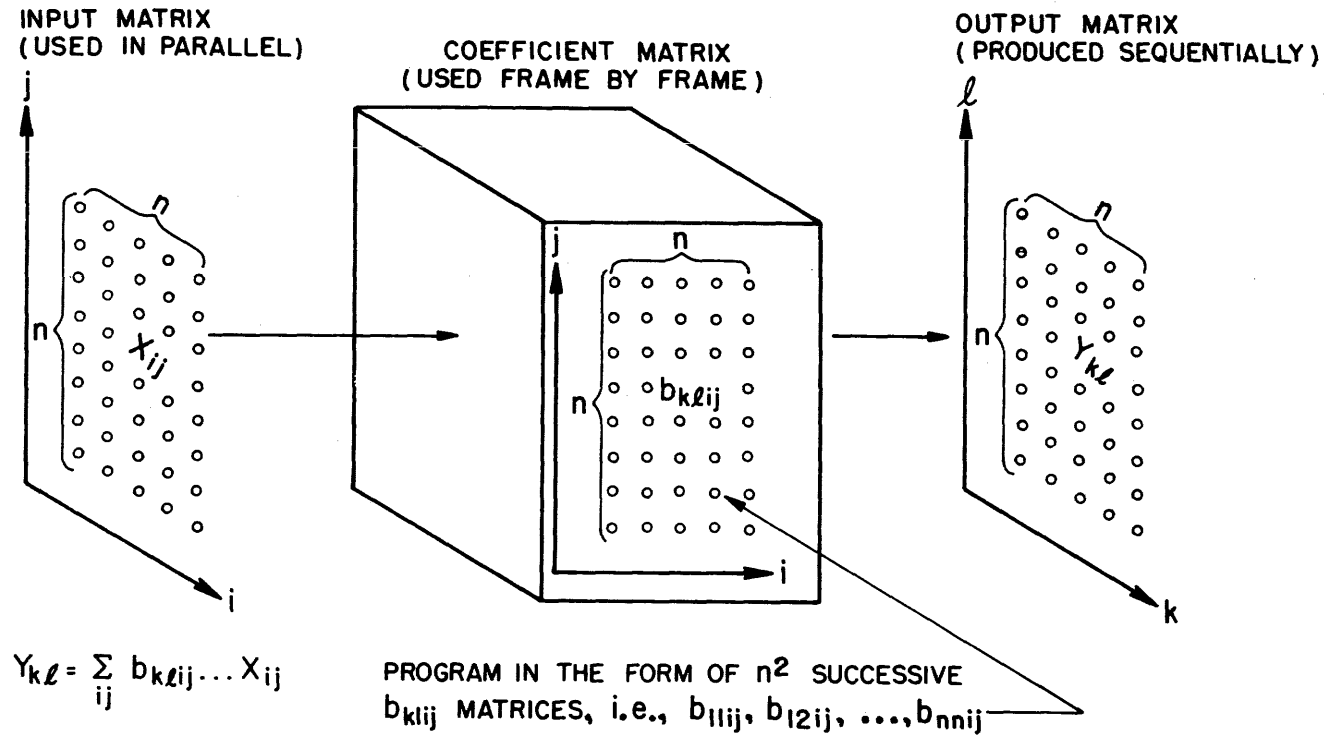


Figure 14 – Transformatrix

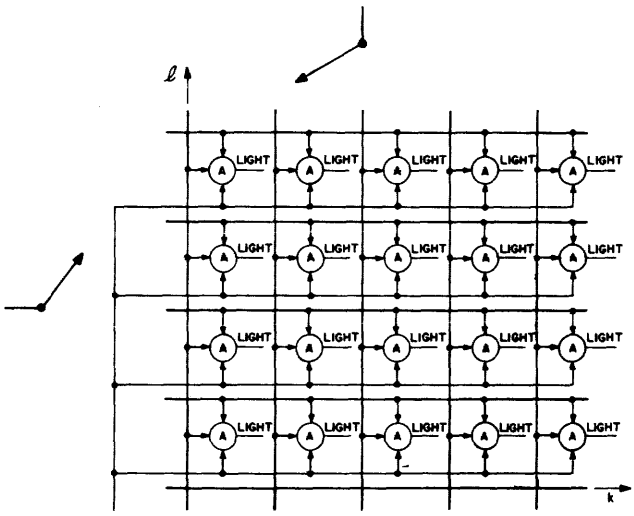


Figure 15 – Scanning matrix for the stochastic image transformer

$$b_{klij} = \delta(k - i)\,\delta(l - j) \qquad (6.2)$$

Similarly it is easy to see that a Fourier Transformation corresponds to

$$y_{kl} = \sum_i \sum_j x_{ij} e^{(ki+lj)\nu\sqrt{-1}} \qquad (6.3)$$

The real and imaginary parts of $e^{(ki+lj)}$ give the b_{klij}'s immediately: The latter will have real and imaginary parts. Short of displaying $(y_{kl})^2$, it might therefore be necessary to split the output matrix into two parts.

REFERENCES

1 S T RIBEIRO
Comments on pulsed-data hybrid computers
IEEE Transactions on Electronic Computers Vol EC-13 October 1964

2 L O GILSTRAP H J COOK C W ARMSTRONG
Study of large neuromime networks
Adaptronics Interim Engineering Report No 1 to the Air Force Avionics Laboratory USAF Wright-Patterson AFB Ohio August 1966

3 C AFUSO
Quarterly technical progress reports
Circuit Research Section Department of Computer Science University of Illinois starting January 1965 to present

4 B R GAINES
Techniques of identification with the stochastic computer
IFAC Symposium 1967 Prague

5 B R GAINES
Stochastic computer thrives on noise
Electronics Vol 40 No 14 July 10 1967 p 72-79

6 S T RIBEIRO
Random-pulse machines
IEEE Transactions on Electronic Computers Vol EC-16 June 1967