Direct expansion of time function: Another convenient expression of the expansion is now presented. With the change of variables,

$$w^{-1} = \frac{1}{\sqrt{b}} \frac{az-b}{az-1} \quad z^{-1} = a\frac{\sqrt{b}w^{-1}-1}{\sqrt{b}w^{-1}-b} \tag{7}$$

we obtain a new representation from the z-transform of eqn. 5:

$$F(w^{-1}) = \frac{\sqrt{b}w^{-1}-b}{1-b} \sum_{n=0}^{N} f_n w^{-n} \tag{8}$$

Thus for analytical functions, the coefficients f_n may be computed by transforming them into a series in $F(w^{-1})$ and dividing by $(\sqrt{b}w^{-1}-b)$. When $F(w^{-1})$ is a rational fraction of w^{-1} it is easier to use a polynomial division which gives the coefficients f_n.

Bilateral decomposition: Consider the identification of a time function defined for negative and positive time by a finite sum of Laguerre functions. We define the bilateral z-transform by

$$F(z) = \sum_{k=-\infty}^{+\infty} f(k) z^{-k}$$

for values of z ensuring the convergence of $F(z)$. We introduce the two causal functions

$$\begin{aligned} f^+(k) &= f(k) && \text{for } k \geq 0 \\ f^+(k) &= 0 && \text{for } k < 0 \\ f^-(k) &= f(-k-1) && \text{for } k \geq 0 \\ f^-(k) &= 0 && \text{for } k < 0 \end{aligned}$$

Their z-transforms are then defined by

$$F^+(z) = \sum_{k=0}^{+\infty} f^+(k) z^{-k} \tag{9a}$$

$$F^-(z) = \sum_{k=0}^{+\infty} f^-(k) z^{-k} \tag{9b}$$

It is then clear that

$$F(z) = F^+(z) + zF^-\left(\frac{1}{z}\right) \tag{10}$$

According to the Laguerre expansion of a signal for positive time, we may write (with different parameters (a_+, b_+) and (a_-, b_-))

$$f^+(k) = \sum_{n=0}^{+\infty} f_n^+ L_n(k, a_+, b_+) \tag{11a}$$

$$f^-(k) = \sum_{n=0}^{+\infty} f_n^- L_n(k, a_-, b_-) \tag{11b}$$

which have z-transforms

$$F^+(k) = \sum_{n=0}^{+\infty} f_n^+ L_n(z, a_+, b_+) \tag{12a}$$

$$F^-(k) = \sum_{n=0}^{+\infty} f_n^- L_n(z, a_-, b_-) \tag{12b}$$

We may now expand the time function bilaterally as

$$F(z) = \sum_{n=0}^{+\infty} f_n^+ \frac{(b_+^{-1/2} - a_+^{-1} b_+^{1/2} z^{-1})^n}{(1-a_+^{-1}z^{-1})^{n+1}} + z\sum_{n=0}^{+\infty} f_n^- \frac{(b_-^{-1/2} - a_-^{-1} b_-^{1/2} z)^n}{(1-a_-^{-1}z)^{n+1}} \tag{13}$$

To simplify $F(z)$, we choose

$$a_+ = a \qquad b_+ = b$$

$$a_- = \frac{b}{a} \qquad b_- = b$$

$$F(z) = \sum_{n=0}^{+\infty} f_n^+ \frac{(b^{-1/2} - a^{-1}b^{1/2}z^{-1})^n}{(1-a^{-1}z^{-1})^{n+1}} - a^{-1}b^{1/2} \sum_{n=0}^{+\infty} f_n^- \frac{(b^{-1/2} - a^{-1}b^{1/2}z^{-1})^{-(n+1)}}{(1-a^{-1}z^{-1})^{-n}}$$

Finally, by defining

$$\begin{aligned} f_n &= f_n^+ && \text{if } n \geq 0 \\ f_{-(n+1)} &= -f_n^- a^{-1} b^{1/2} && \text{if } n < 0 \end{aligned}$$

the z-transform is written as

$$F(z) = \frac{1}{1-a^{-1}z^{-1}} \sum_{n=-\infty}^{+\infty} f_n \left(\frac{b^{-1/2} - a^{-1}b^{1/2}z^{-1}}{1-a^{-1}z^{-1}}\right)^n \tag{14}$$

Simplification occurs when the change of variables

$$w^{-1} = \frac{b^{-1/2} - a^{-1}b^{1/2}z^{-1}}{1-a^{-1}z^{-1}} \tag{15}$$

is substituted into eqn. 14, giving

$$F(w^{-1}) = \frac{\sqrt{b}w^{-1}-b}{1-b} \sum_{n=-\infty}^{+\infty} f_n w^{-n} \tag{16}$$

Summarising, the computation of the coefficients f_n may be achieved using different techniques. For an analytical time function defined on the interval $[0, +\infty]$, we may use eqn. 8, for experimental data on the interval $[0, K]$ eqn. 6 is convenient, and for time functions defined in negative and positive time, bilinear decomposition (eqn. 16) must be applied.

Conclusion: We have shown how to expand a discrete time series using the discrete Laguerre functions. The main contribution of the proposed method is the extension of a classical approach to the case involving series defined over a interval defined in negative and positive time. Application of these expansions may be developed in the field of signal modelling and system identification where the input and the output of the concerned process are defined in an interval including the time origin. This occurs in anticipative or noncausal system applications such as image processing.

25 April 1995
Electronics Letters Online No: 19950663

S. Ekongolo, D. Maquin and J. Ragot (*Centre de Recherche en Automatique de Nancy - CNRS UA 821, Institut National Polytechnique de Lorraine, 2, Avenue de la Forêt de Haye, F-54516 Vandoeuvre, France*)

References

1 WAHLBERG, B.: 'System identification using Laguerre models', *IEEE Trans. Autom. Control*, 1991, **36**, (5), pp. 551–562
2 MÄKILÄ, P.M.: 'Laguerre series approximation of infinite dimensional systems', *Automatica*, 1990, **26**, (2), pp. 985–995
3 MARMONIER, R., ROESCH, M., and RAGOT, J.: 'Some methods of decomposition of a time function into discrete Laguerre function', *J. A*, 1976, **17**, (4), pp. 215–218

Compact digital pseudorandom number generator

D. Anguita, S. Rovetta and R. Zunino

Indexing term: Random number generation

A general-purpose easy-to-implement random number generator is presented. It features an 8 bit word size, good statistical properties and repeatability of the generated sequence. The circuit has been included in the design of many systems, ranging from neural networks to cryptography.

Introduction: Many applications require the use of a random number generator with particular properties [1]. Such a device often lies at the very heart of a system. Examples of applications in the electronic field include cryptography (for which the period length serves as the minimum protection against brute force attacks) and initialisation of parameters for neural networks train-

ing. We require that the system generate a sequence of numbers with good statistical properties, i.e. as uniform as possible. In the case of cryptography, reproducibility is also crucial. In these situations, a pseudorandom generator is used instead of a real noise source. A pseudorandom generator starts from a single number (the seed), and creates a sequence by a chaotic process simulating randomness. We aim to obtain the longest possible period before the repetition of a sequence. Moreover, when the generator algorithm is planned to be implemented in hardware, the simplest circuit is obviously preferred.

The proposed scheme shows a good statistical behaviour, easy scalability to increase the length of the period, a very simple realisation in digital hardware, and the ability to produce zero as part of the output sequence. This feature allows the implementation of a random variable with values in [0, *MAX*] as often assumed in some applications, without requiring additional computations that would reduce the compactness of the circuit.

Pseudorandom number generators: The proposed generator is based on the model of linear recursive generation (LRG) [2], particularly well-suited for hardware implementation. The basic building block is a shift register. The method of LRG is based on a linear recursive sequence of bits, defined as follows:

$$X_k = C_1 X_{k-1} + C_2 X_{k-2} + \ldots + C_p X_{k-p}$$

where X_k is the kth bit of the generated sequence, and the p coefficients C_i are binary constants that determine the behaviour of the generator. The linear recursive sequence is initialised with a set of p values $\{X_1, \ldots, X_p\}$. The first p–1 constants C_i take on values in the range $\{0, 1\}$ and $C_p = 1$. As the next bit X_k depends only on the past p bits, the maximum period n we can achieve will be $n = 2^p$–1. The following holds [1]:

Theorem: The period n has its maximum possible value 2^p–1 if and only if the polynomial

$$f(x) = 1 + C_1 x + C_2 x^2 + \ldots + C_p x^p$$

is prime over the field of the polynomials with coefficients in {1, 0}. The sequence generated by such a set of coefficients is statistically balanced (its average is $\Sigma_k X_k/n = 1/2$).

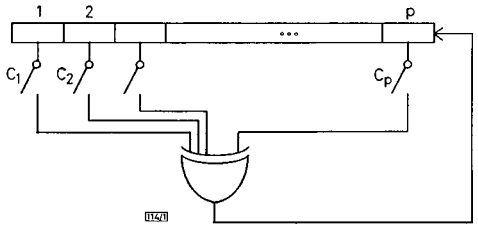

Fig. 1 *Linear recursive generator*

The circuit implementation of such a simple generator, shown in Fig. 1, is straightforward. It involves only one shift register and an exclusive-or computing the sum of the p bits $C_1 X_{k-1}, \ldots, C_p X_{k-p}$. This basic scheme produces a single bit stream with pseudorandom behaviour, but can also be used to build up a generator for a stream of multibit words. If a word length of b bits is required, the first b bits of the shift register can be used (if $b < p$), or another shift register of suitable length can be appended to the output of the single-bit circuit to accumulate enough bits to form a complete word. In either case, the period of the sequence is reduced to $(2^p–1)/b$ words. A better generator can be designed by using b parallel LRGs. A shift matrix $\mathbf{C}_{ij}$ is obtained by considering the b rows of p shift coefficients each. It can be shown that, if the shift matrix has full rank, the period of this circuit is still 2^p–1. A more complex realisation is the main drawback of this solution.

Usually, a sequence length of 255 is still not sufficient for most realisations, even for those requiring 8 bit words. Hence the basic LRG should be improved by lengthening its period. It is not very difficult to achieve such a goal, but we must also satisfy the important requirement of statistical uniformity (the sequence should be characterised by impulsive autocorrelation). When designing the generator, we must also take into account the actual word size b required. If analysed statistically, some schemes behave better when realised with a larger number of bits. In general, it is harder to design a generator with good independence properties if the word size is smaller. For these reasons, once a generator has been designed, it is a good practice to simulate it in software and to analyse its output by means of a statistical independence test, like the standard chi-squared test.

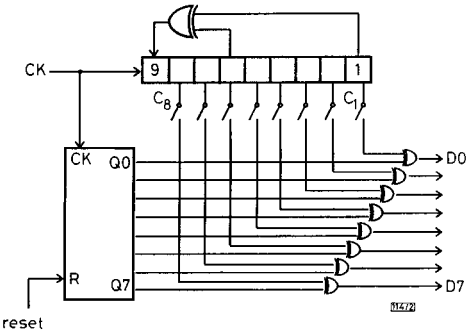

Fig. 2 *Proposed circuit*

Implementation: The proposed circuit is shown in Fig. 2. It is based on the idea of combining two generator blocks in such a way that their periods interact to form the maximum possible sequence length. This means that, if the periods of the two blocks are n_1 and n_2, respectively, the combination should yield a period $n = n_1 n_2$. This property can be achieved when the two periods are mutually prime.

The first block is an LRG with 9 bits, initialised with an 8 bit seed. The last bit is loaded with a fixed value (selected at design time). The period is thus 2^9–1 = 511.

If the constant initialisation bit is set to 1, the generated sequence also includes the value 0, which is consequently a legitimate value for the seed. This property may be required, for instance, in all applications in which the seed is user-selectable, and hence it can not be ensured that a particular value will never be introduced.

The second block need not be another pseudorandom generator but only an element introducing a decorrelation among the successive bits of the sequence; to this end, it is sufficient to provide an 8 bit counter with a period 256. As 511 and 256 are mutually prime, the global period of the generator will amount to 256 × 511 = 130816.

The initialisation is performed with two 8 bit words, one to set up the counter and the other to provide a seed for the LRG.

The standard χ^2 test yielded good results in terms of sequence uniformity. The parameter χ^2 remained within acceptable limits for most of the sequences, generated with different seeds.

Conclusion: The proposed circuit has been used in the design of many systems currently under development. Some examples include the VLSI implementation of a vector quantisation neural network requiring a set of initial random values for the training process (plastic neural gas [3]), and a hardware cryptographic system implementing a scaleable fractal algorithm. A more exhaustive validation of the statistical properties of a pseudorandom sequence would be needed, especially for cryptographic applications, but the definition of a theoretically sound and practically useful testing procedure is still an open problem.

The proposed circuit is especially well suited to applications requiring a compact circuit to generate random sequences, as in digital realisations of distributed computing applications [4].

Acknowledgment: The authors thank M. Cappelli, from whose ideas this work originated .

27 April 1995
Electronics Letters Online No: 19950695

D. Anguita, S. Rovetta and R. Zunino (*D.I.B.E., University of Genoa, Via All'Opera Pia 11a, I-16145 Genova, Italy*)

References

1 L'ECUYER, P.: 'Uniform random number generation', *Ann. Oper. Res.*, 1994, **53**, pp. 77–120
2 KNUTH, D.E.: 'The art of computer programming: Seminumerical algorithms (Addison-Wesley, 1981), 2nd Edn., Vol. 2
3 RIDELLA, S., ROVETTA, S., and ZUNINO, R.: 'Plastic neural gas for adaptive vector quantisation', submitted to *IEEE Trans. Neural Netw.*
4 PARODI, G.C., RIDELLA, S., and ZUNINO, R.: 'Using chaos to generate keys for associative noise-like coding memories', *Neural Net.*, 1993, **6**, (4), pp. 559–572

Digit serial division algorithm

A.E. Bashagha and M.K. Ibrahim

Indexing term: Digital arithmetic

A new radix digit serial division algorithm is presented. The speed of the proposed algorithm is nearly twice that of an existing design. Moreover, the new algorithm requires less area for a digit size of up to 8bits (for a 32bit quotient).

Introduction: The digit serial approach has recently been developed as a compromise between the fast and expensive bit parallel approach and the slow and cheap bit serial realisation [1]. The digit serial structure processes a number of bits n, called a digit at one cycle, where n varies between 1 bit and the wordlength, N. Digit serial notation is also used in the context of redundant number based systems which are also known as on-line arithmetic systems [2]. The drawback of the on-line systems is an increased size of the computational elements and the overhead of data conversion.

New digit serial algorithms and architectures based on radix-2^n arithmetic have recently been developed [3, 4]. Each quotient bit is generated in m cycles, where m is the number of radix-2^n digits of the word. The throughput rate of these architectures can be increased by pipelining the architecture to the digit level [3] or even the bit level [4]. However, the pipelining of the architecture will increase the latency and the number of latches (i.e. the area). We propose a modified version of the original digit serial algorithm of [3, 4], where k quotient bits can be generated in $m+k$ cycles. This results in a reduction of the computation time per quotient bit and, moreover, a reduction in area since fewer latches are required.

Digit serial division: We consider the division process $Q = A/D$, where the dividend A, the divisor D, and the quotient Q, are $2N$ bit, N bit, and N bit fixed point fractions, respectively. It is assumed that $|A| < |D|$ and $|D| \neq 0$. The input operands, A and D, are divided into $2m$ digits and m digits, respectively, where each digit consists of n bits (i.e. $N = mn$). They are fed in a digit serial form (digit-by-digit) starting from the least significant digits (LSDs). The N bit quotient, Q, is generated in a bit parallel form and can be converted to a digit serial form. The m-digit partial remainder (hereafter referred to as the remainder) PR_i, for $i = 1, ..., N$ is generated in a digit serial form starting from the LSD. It is assumed that all the data, A, D, Q, and PR_i are in two's complement form.

In the original digit serial division algorithm [3, 4], m cycles are required to generate each quotient bit q_i, for $i = 1, ..., N$. This bit is used to control the $(i+1)$th step of the algorithm such that an addition (subtraction) is carried out if $q_i = 0$ (1). Therefore, the addition (subtraction) operation of the $(i+1)$th step cannot be started until the quotient bit q_i of the ith step is generated. The delay between feeding the LSDs of the inputs at the ith step and generating q_i is m cycles. Therefore, the LSD (and also the other digits) of the ith remainder has to be delayed by m cycles before being processed in the next step. The throughput rate of this algorithm is one quotient bit every m cycles, and the latency is mN cycles. The throughput rate can be increased by pipelining the architecture to the bit level [4], but the area will increase and the latency will become mnN (i.e. N^2) cycles.

Proposed algorithm: As indicated earlier, the value of q_i in the ith step is either 0 or 1, and hence the operation in the $(i+1)$th step is either addition or subtraction. In the new algorithm, instead of waiting m cycles until q_i is generated, we start the $(i+1)$th step and use the generated digits of the remainder PR_i without any additional delay. Since q_i is not yet known, we carry out both operations in the $(i+1)$th step, i.e. addition and subtraction. The addition and subtraction are carried out in parallel, and two possible values of the remainder, PR_{i+1} at the $(i+1)$th step, are generated. Once the quotient bit q_i is obtained, it is used as a control mode to an n bit multiplexer to select one of the two possible values of the remainder, PR_{i+1}, such that $PR_{i+1} = 2PR_i+D$ if $q_i = 0$ and $PR_{i+1} = 2PR_i-D$ if $q_i = 1$. Therefore, the ith and the $(i+1)$th steps are overlapped, and the two steps are carried out in $m+1$ cycles instead of $2m$ cycles as in the original algorithm. This procedure can be generalised such that k steps are overlapped together to generate k quotient bits in $m+k$ cycles.

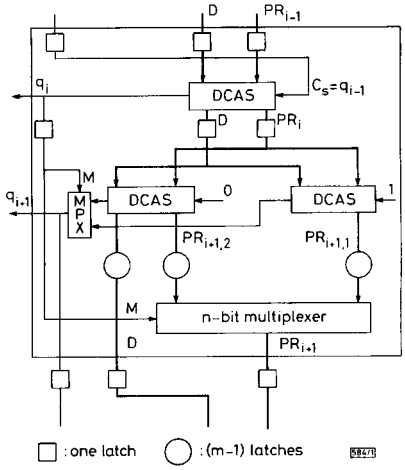

Fig. 1 *Basic digit serial cell*

Digit serial architecture: The basic cell used to implement this algorithm is shown in Fig. 1. For simplicity, we select $k = 2$ such that two quotient bits are generated every $m+1$ cycles. In this Figure, DCAS is an n bit digital controlled add/subtract cell [4], $PR_{i+1,1}$ and $PR_{i+1,2}$ are the possible values of the $(i+1)$th remainder, and PR_{i+1} is the correct value (i.e. PR_{i+1} equals either $PR_{i+1,1}$ or $PR_{i+1,2}$). The n bit multiplexer has two n bit inputs (one digit of $PR_{i+1,1}$ and the corresponding digit of $PR_{i+1,2}$) and one n bit output (one digit of PR_{i+1}) MPX is a 2 bit to 1 bit multiplexer used to select the quotient bit q_{i+1}. Both multiplexers are controlled by a control mode M (where $M = q_i$). The control signal C_s for the first DCAS cell equals q_{i-1}.

The area per quotient bit of the proposed structure A_p and the corresponding time T_p are now compared with the area A_0 and time T_0 of the original algorithm [3, 4] using the figures of merit in [5], with the area and time being given in units of two-input NAND gate. We also assume that the proposed architecture and that of [3, 4] are pipelined to the digit level. It should be noted that A_p and T_p are calculated as half the area and time, respectively, required to generate q_i and q_{i+1} as shown in the box of Fig. 1. The expressions for A_p, T_p, A_0, and T_0 and the corresponding values for $k = 2$ and $N = 32$ are as follows:

$$A_p = 1.5A_{DCAS} + (1.5mn + 0.5n + 1)A_{latch} + 0.5(n+1)A_{MPX} \qquad A_p = 355 + 23.5n$$

$$T_p = 0.5(m+1)T_{DCAS} \qquad T_p = 85 + 5m + 2.5n$$

$$A_o = A_{DCAS} + 2mnA_{latch} \qquad A_o = 455 + 12n$$

$$T_o = mT_{DCAS} \qquad T_o = 160 + 10m$$

where FA is a full adder, XOR is an exclusive-or gate, and MPX is a 2 bit to 1 bit multiplexer. It is assumed that a carry propagate adder is used in the DCAS cell, and hence the area of the DCAS cell is that of an n(FA+XOR)+latch, whereas the corresponding time is that of an nFA+XOR+latch. The area ratio A_p/A_0 and the