

## 10 Lehmer Random Number Generator. A case study

- Fast random number generators are essential blocks of many signal processing and modelling algorithms.
- We discuss a number of possible hardware implementations of a 31-bit Lehmer Random Number Generator (LRNG).
- Some of these implementations were described in [1] and the reader is referred to this work for further details not presented in this case study.
- The Lehmer random number generator was used in some software packages, specifically in [2], and the idea here is to design an integrated circuit implementing the LRNG, or embed it in a bigger design, in order to speed up the software applications.
- We will show that generation of the next (pseudo-) random numbers by LRNG can be reduced to addition/subtraction of not more than seven appropriately rotated copies of the current random number modulo a specific Mersenne prime number.
- Such operation can be performed in at least three different ways, namely: word-parallel, word-serial and bit-serial. These three ways are typical to all arithmetic algorithms.
- This aspect makes the LRNG case study a good representative of a big class of signal processing algorithms.

## 10.1 Description of Lehmar random number generator

The Lehmer random number generator belongs to the class of the linear multiplicative congruential generator.

It generates 31-bit uniformly distributed pseudo-random numbers.

The main part of the generator, NRN, produces the **next random number**,  $R$ , from the current random number,  $Z$ , as in Figure 10–1.

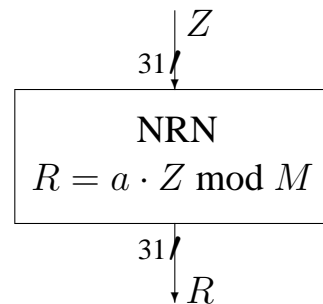


Figure 10–1: The Next Random Number (NRN) block of the Lehmer random number generator.

The algorithm can be described by the following multiplicative congruential expression:

$$\begin{aligned}
 R &= a \cdot Z \bmod M \\
 (R \text{ and } a \cdot Z \text{ are congruent modulo } M)
 \end{aligned}
 \tag{10.1}$$

(Note that we have

$$\frac{a \cdot Z}{M} = Q + \frac{R}{M}$$

)

$$R = a \cdot Z \bmod M$$

where

- $Z$  and  $R$  are the **current** and the **next** random numbers, respectively,
- $a$  is a specially selected 15-bit **multiplier**:

$$a = 7^5 = 16807 = (100\ 0001\ 1010\ 0111)_2$$

with seven ones located in the positions: 14, 8, 7, 5, 2, 1, 0,

- $M$  is the **modulus**:

$$M = 2^{31} - 1$$

which is a 31-bit Mersenne prime number.

It can be shown that using eqn (10.1) recursively, and starting with any seed number

$$Z \in [1 \dots M - 1]$$

we will generate  $M - 2$  different numbers.

At every step generated numbers are uniformly distributed in the range  $R \in [1 \dots M - 1]$ .

If we interpret generated pseudo-random numbers as fractions, then we have:

$$R \cdot 2^{-31} \in [2^{-31} \dots 1 - 2^{-30}]$$

## 10.2 The implementation method

In the subsequent sections we will show that the LRNG algorithm to be implemented by the NRN circuit and described by eqn (10.1) can be reduced to the following operations:

$$R = (E_{14} + E_8 + E_7 + E_5 + E_2 + E_1 + E_0) \bmod M \quad (10.2)$$

$$= (E_{14} + E_8 + E_7 + E_5 + E_3 - E_0) \bmod M \quad (10.3)$$

where

$$E_i = \text{lrot}(i, Z)$$

is the left rotation of the 31-bit input number  $Z$

$$Z = z_{30}z_{29} \dots z_{31}z_0$$

by  $i$  positions. Obviously we have  $E_0 = Z$ . For future reference, the rotated input numbers are collected in the following matrix:

$$\begin{bmatrix} E_{14} \\ E_8 \\ E_7 \\ E_5 \\ E_2 \\ E_1 \\ E_0 \end{bmatrix} = \begin{bmatrix} z_{16} & z_{15} & z_{14} & \dots & z_0 & z_{30} & \dots & z_{24} & z_{23} & z_{22} & z_{21} & z_{20} & z_{19} & z_{18} & z_{17} \\ z_{22} & z_{21} & z_{20} & \dots & & z_{30} & z_{29} & z_{28} & z_{27} & z_{26} & z_{25} & z_{24} & z_{23} & & \\ z_{23} & z_{22} & z_{21} & \dots & & z_0 & z_{30} & z_{29} & z_{28} & z_{27} & z_{26} & z_{25} & z_{24} & & \\ z_{25} & z_{24} & z_{23} & \dots & & z_2 & z_1 & z_0 & z_{30} & z_{29} & z_{28} & z_{27} & z_{26} & & \\ z_{28} & z_{27} & z_{26} & \dots & & z_5 & z_4 & z_3 & z_2 & z_1 & z_0 & z_{30} & z_{29} & & \\ z_{29} & z_{28} & z_{27} & \dots & & z_6 & z_5 & z_4 & z_3 & z_2 & z_1 & z_0 & z_{30} & & \\ z_{30} & z_{29} & z_{28} & \dots & & z_7 & z_6 & z_5 & z_4 & z_3 & z_2 & z_1 & z_0 & & \end{bmatrix} \quad (10.4)$$

Hence, the **LRNG algorithm** can be implemented by either

- **adding seven** appropriately rotated 31-bit numbers modulo  $M$  as in eqn (10.2),

$$R = (E_{14} + E_8 + E_7 + E_5 + E_2 + E_1 + E_0) \bmod M$$

or

- **adding five and subtracting one** number as in eqn (10.3)

$$R = (E_{14} + E_8 + E_7 + E_5 + E_3 - E_0) \bmod M$$

We will show that addition modulo  $M = 2^{31} - 1$  can be easily performed using a **cyclic carry method**, that is, adding the output carry back to the least significant position.

### 10.3 Basic properties of the modulo operation

In what follows we need to use a few basic properties of the modulo operations. We start with the following definition:

$$\begin{aligned} A = B \bmod M \quad (&A \text{ and } B \text{ are congruent modulo } M) \\ \iff \text{there exists an integer } q \text{ such that } &B = q \cdot M + A \end{aligned} \tag{10.5}$$

In other words  $A$  is a remainder from division of  $B$  by  $M$ .

#### Example

$$19 \bmod 7 = (2 \cdot 7 + 5) \bmod 7 = 5 \quad (5 \text{ and } 19 \text{ are congruent modulo } 7)$$

$$12 \bmod 7 = (1 \cdot 7 + 5) \bmod 7 = 5 \quad (5 \text{ and } 12 \text{ are congruent modulo } 7)$$

From the definition (10.5) we can easily prove the following two useful properties:

**Property 1:** For any integer  $q$  we have

$$(q \cdot M + r) \bmod M = r \bmod M \tag{10.6}$$

**Property 2:** Addition modulo  $M$  is distributive, that is:

$$(A + B) \bmod M = (A \bmod M + B \bmod M) \bmod M \tag{10.7}$$

Example:

$$(11 + 8) \bmod 7 = (11 \bmod 7 + 8 \bmod 7) \bmod 7 = (4 + 1) \bmod 7$$

## 10.4 Derivation and transformations of the LRNG algorithm

Using the above two properties of the modulo operation, we can decompose the algorithm (10.1) into the following sum:

$$\begin{aligned}
 R &= a \cdot Z \bmod M = (100\ 0001\ 1010\ 0111)_2 \cdot Z \bmod M \\
 &= (Z \cdot 2^{14} + Z \cdot 2^8 + Z \cdot 2^7 + Z \cdot 2^5 + Z \cdot 2^2 + Z \cdot 2 + Z) \bmod M \\
 &= (E_{14} + E_8 + E_7 + E_5 + E_2 + E_1 + E_0) \bmod M
 \end{aligned} \tag{10.8}$$

where

$$E_i = Z \cdot 2^i \bmod M \tag{10.9}$$

The method of **evaluating**  $E_i$  defined in eqn (10.9) is graphically depicted in Figure 10–2. We start with

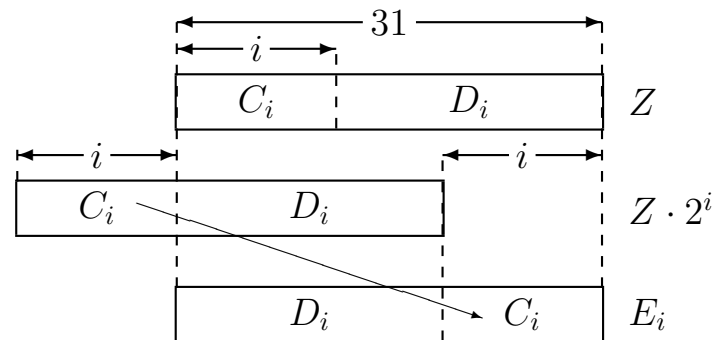


Figure 10–2: Graphical illustration of formation of  $E_i$

partitioning the 31-bit input number  $Z$  into an  $i$ -bit more significant part and the  $(31 - i)$ -bit less significant part, so that we have:

$$Z = C_i \cdot 2^{31-i} + D_i$$

After shifting  $Z$  by  $i$  positions we have

$$Z \cdot 2^i = C_i \cdot 2^{31} + D_i \cdot 2^i$$

Now, performing the modulo  $M$  operation and applying properties (10.6) and (10.7), we can evaluate  $E_i$  in the following way:

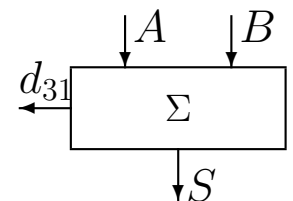
$$\begin{aligned} E_i &= Z \cdot 2^i \bmod M = (C_i \cdot 2^{31} + D_i \cdot 2^i) \bmod M \\ &= (C_i \cdot 2^{31} - C_i + C_i + D_i \cdot 2^i) \bmod M = (C_i \cdot (2^{31} - 1) + D_i \cdot 2^i + C_i) \bmod M \\ &= (C_i \cdot M + D_i \cdot 2^i + C_i) \bmod M \end{aligned}$$

and finally, after dropping out  $C_i \cdot M$ , we have

$$E_i = (D_i \cdot 2^i + C_i) \bmod M = \text{lrot}(i, Z) \quad (10.10)$$

which shows that  $E_i$  can be indeed created by the left  $i$ -position rotation of the input number  $Z$ .

In order to consider a method of **addition modulo**  $M$  we first observe that a standard addition or subtraction of two 31-bit numbers can be represented by the following equation:



$$A \pm B = d_{31} \cdot 2^{31} + S \quad (10.11)$$

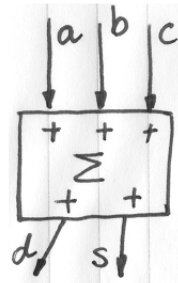
where  $S$  is the 31-bit sum and  $d_{31}$  is the output carry.

In general, for addition  $d_{31} \in \{+1, 0\}$ , and for subtraction  $d_{31} \in \{0, -1\}$ .



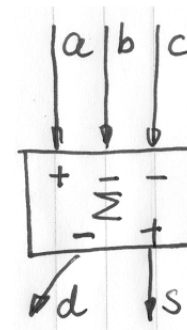
1-bit adder  
 $a+b+c = 2d+s$

+ + +			#	+2 +	
c	b	a		d	s
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	3	1	1
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1



1-bit subtracter  
 $a-b-c = -2d+s$

- - +			#	-2 +	
c	b	a		d	s
0	0	0	0	0	0
0	0	1	+1	0	1
0	1	0	-1	1	1
0	1	1	0	0	0
1	0	0	-1	1	1
1	0	1	0	0	0
1	1	0	-2	1	0
1	1	1	-1	1	1

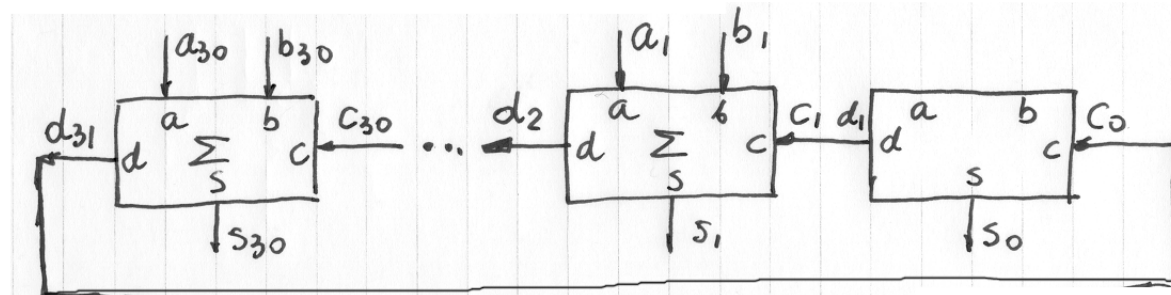


$$s = a \oplus b \oplus c$$

$$d = ab + bc + ca$$

$$d = \bar{a}b + bc + c\bar{a}$$

A CARRY PROPAGATE ADDER/SUBTRACTER MOD M



Now, performing the modulo  $M$  operation and applying properties (10.6) and (10.7), we can obtain the following 31-bit result  $F$ :

$$\begin{aligned} F &= (A \pm B) \bmod M = (d_{31} \cdot 2^{31} + S) \bmod M = (d_{31} \cdot 2^{31} - d_{31} + d_{31} + S) \bmod M \\ &= (d_{31} \cdot M + S + d_{31}) \bmod M \end{aligned}$$

and finally we have

$$F = (A \pm B) \bmod M = S + d_{31} \quad (10.12)$$

This is an elegant result which states that addition/subtraction modulo  $M = 2^{31} - 1$  is equivalent to a **cyclic addition** in which the output carry from the most significant position is fed back and added to the least significant position as illustrated in Figure 10–3.

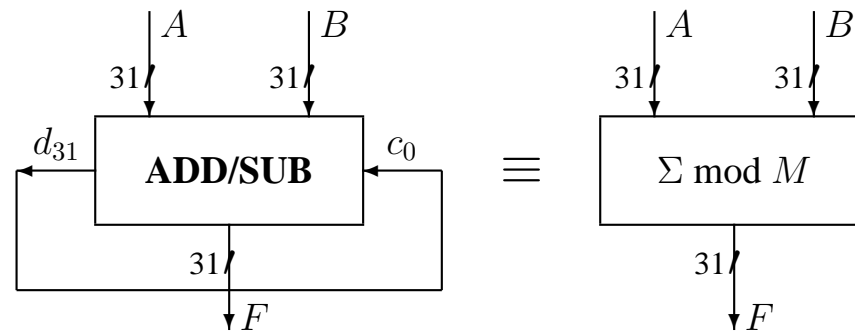


Figure 10–3: The cyclic addition/subtraction of two 31-bit numbers.

It can be easily shown that feeding back the output carry does not result in an unstable circuit in which the output carry oscillates between 0 and 1. Note that the output carry  $d_{31} = 1$  is generated by the most significant position with bits  $a_i$  and  $b_i$  being both equal to 1 irrespective of the input carry to the  $i$ -th position. This fact makes oscillations impossible.



## 10.5 Word-parallel implementation of the LRNG

The word-parallel implementation of the Next Random Number (NRN) circuit is a structured combinational circuit consisting of a number of appropriately connected adders and subtractors to add seven or six numbers as in eqn (10.2) or (10.3).

The first approach to build a word-parallel LRNG is presented in the left-hand side of Figure 10–4. This implementation is based on six serially connected 31-bit **carry-propagate** modulo  $M$  adders such as presented in Figure 10–3. The adders add seven numbers modulo  $M$  generating the next 31-bit random number  $R$ .

This solution has two drawbacks: firstly, that adders are connected serially, and secondly, that carry-propagate adders are used. The adding time in the worst case situation is

$$t_1 = 6 \times 31 \times t_a = 186 t_a$$

where  $t_a$  is the propagation time for a single 1-bit adder. For simplicity, we assumed that the carry propagates serially through the 31-bit adder.

An improvement to this solution is presented in the right-hand side of Figure 10–4. Here, the six adders are connected in a form of a tree (the Wallace tree) and the number of adder levels is reduced to three. Hence the maximum value of the adding time has been reduced to

$$t_2 = 3 \times 31 \times t_a = 93 t_a$$

Some improvements can be made by speeding up the carry-propagate circuit. If we assume that the speeding factor is four, we can reduce the NRN formation time to  $46 t_a$  and  $23 t_a$ , respectively.

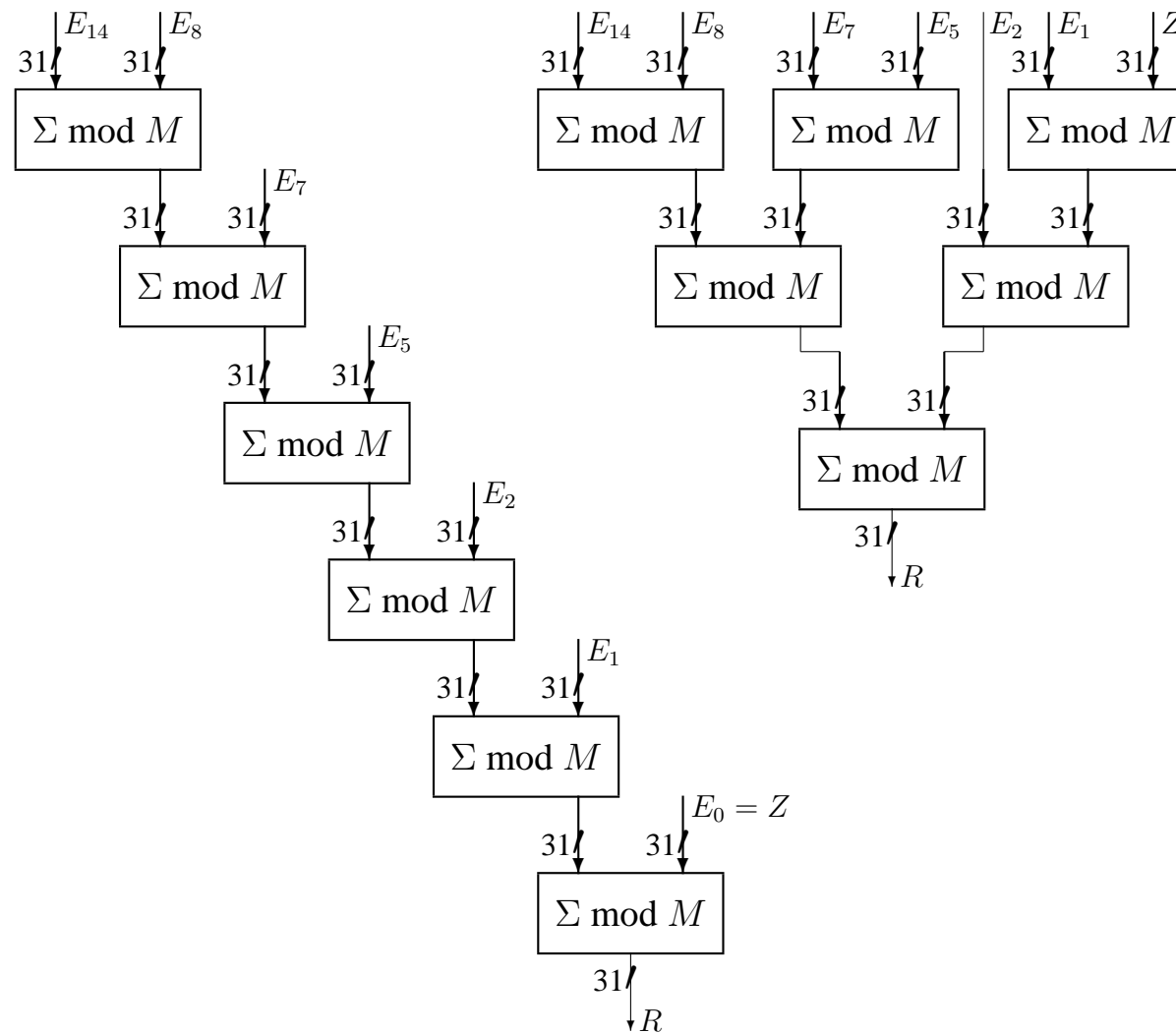


Figure 10–4: Word-parallel implementation of the NRN circuit based on the carry-propagate adders.

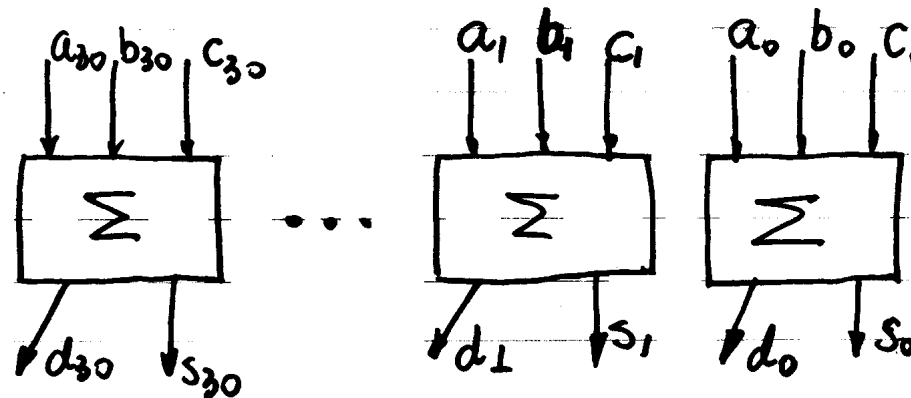
Radical improvement in the adding time is achieved by replacing the carry-propagate adders with the carry-save adders.

The 31-bit **carry-save adder** (CSA) is built from 31 standard 3-input 2-output adders as discussed previously. For convenience we repeat here an arithmetic relationship linking inputs and outputs of the 1-bit adder:

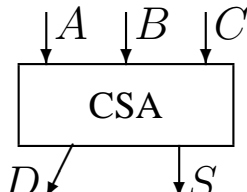
$$\begin{array}{c}
 \downarrow a_i \downarrow b_i \downarrow c_i \\
 \boxed{\Sigma} \\
 \swarrow d_i \quad \downarrow s_i
 \end{array}
 \quad 2 \cdot d_i + s_i = a_i + b_i + c_i \quad (10.13)$$

where  $d_i$  and  $s_i$  are the output carry and sum, respectively,  $a_i, b_i, c_i$  being three inputs.

In the **n-bit carry-save** adder the carry bits generated at the individual positions are not transferred to the next position as in the carry-propagate adder, but instead they directly form the second output.



Therefore, the input and output n-bit words are related in a way as in the 1-bit adder, namely:



$$2 \cdot D + S = A + B + C \quad (10.14)$$

Comparing the carry-save adder described by eqn 10.14 with its carry-propagate counterpart described by eqn 10.11 we note that

- the carry-propagate adder adds **two** n-bit numbers and generates an **n-bit sum** and a **1-bit output carry**,
- the carry-save adder adds **three** n-bit numbers and generates an **n-bit sum** and an **n-bit output carry**.  
Note that because the carry is not propagated between position the result is ready after the time  $t_a$ .

The 31-bit **carry-save adder modulo M** can be easily obtained from the standard carry-save adder in the following way:

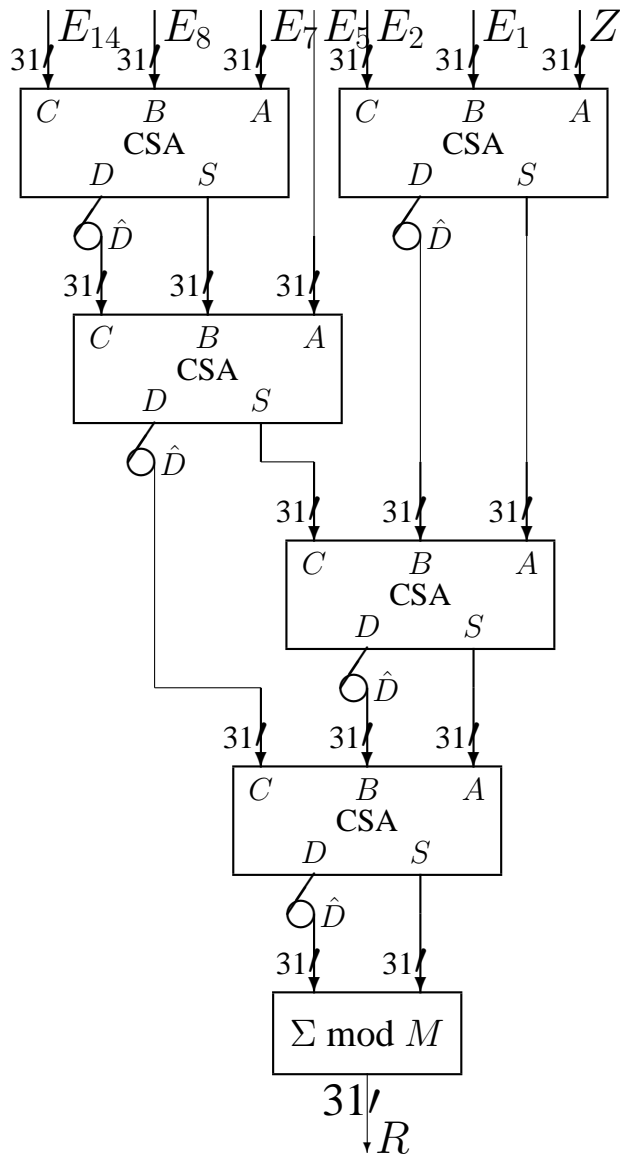
$$(A + B + C) \bmod M = (2 \cdot D + S) \bmod M = S + \hat{D} \quad (10.15)$$

where

$$\hat{D} = 2 \cdot D \bmod M = (d_{29} d_{29} \dots d_0 d_{30})_2 \quad (10.16)$$

In order to perform the modulo  $M$  carry-save addition, the most significant carry bit,  $d_{30}$  must be rotated over to the least significant position, as for the carry-propagate addition.

In other words the modulo  $M$  addition requires no modification to the CSA adder and only a rearrangement of the carry bits is needed to obtain the correct result.



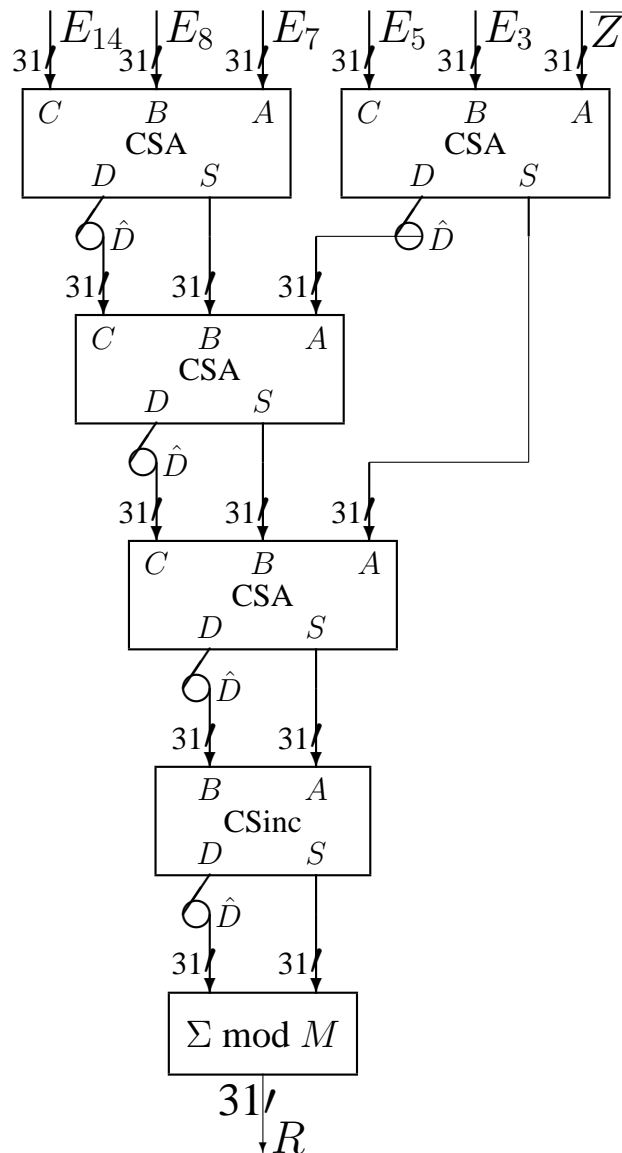
- The word-parallel NRN circuit based on a tree of the modulo  $M$  carry-save adders.
- This implementation combines carry-save adders arranged in a tree as in Figure 10.5.
- The circle on the carry output,  $\hat{D}$ , symbolizes the left rotation operation as described in eqn (10.16).
- The NRN circuit consists of 4 levels of the carry-save adders modulo  $M$  and a final carry-propagate adder which generates the next random number,  $R$ .
- The nominal time to obtain the result is now:

$$t_3 = (4 + 31/4) \times t_a \approx 12t_a$$

which is a significant improvement on the carry-propagate solution.

- The question remains whether application of eqn (10.3) can deliver a better implementation to the one discussed above. Subtraction of one number requires an additional circuit.





- The simplest solution seems to be the one based on the following modification of eqn (10.3) in which the subtraction is replaced with complementation and an increment ( $-E_0 = \overline{E_0} + 1$ ):

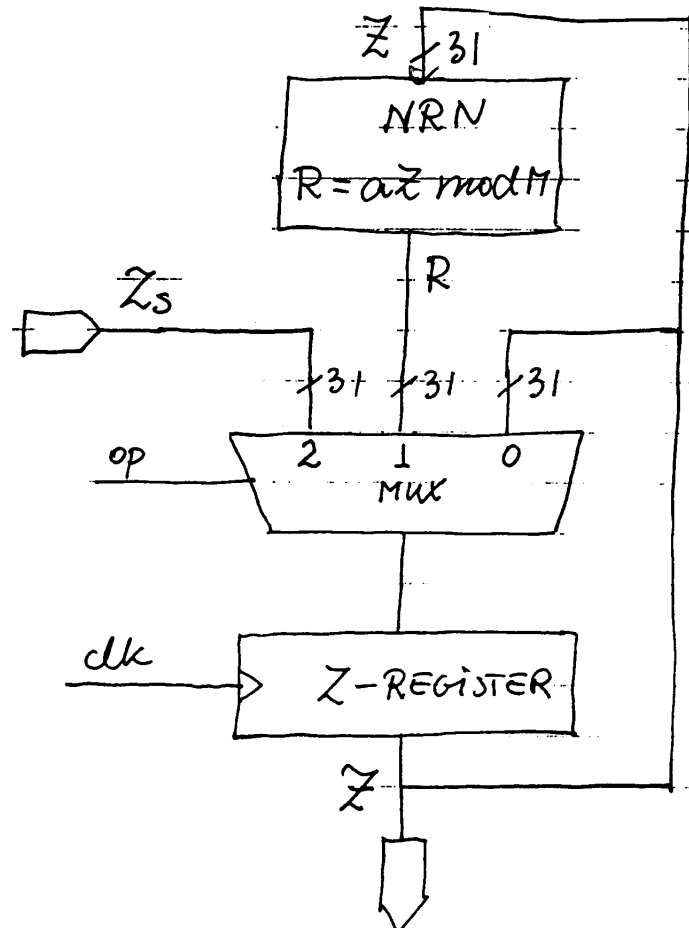
$$R = (E_{14} + E_8 + E_7 + E_5 + E_3 + \overline{E_0} + 1) \bmod M \quad (10.17)$$

- The block-diagram of the NRN circuit based on the above equation is similar to that presented before, however,
- one carry-save adder has been replaced by a carry-save incrementer, CSinc.
- Logic equations for such an incrementer are as follows:

$$s_i = \begin{cases} \overline{a_i \oplus b_i} & \text{for } i = 0 \\ a_i \oplus b_i & \text{for } i = 1 \dots 30 \end{cases} \quad d_i = \begin{cases} a_i + b_i & \text{for } i = 0 \\ a_i \cdot b_i & \text{for } i = 1 \dots 30 \end{cases} \quad (10.18)$$

## 10.6 Block-diagram of the complete generator

The complete generator consists of the Next-Random-Number circuitry as described previously, a register storing the current random number,  $Z$ , and a multiplexer to load the seed (an initial random number).



Operation table

op	operation	
0	$Z \leftarrow Z$	nop
1	$Z \leftarrow (a \cdot Z) \bmod M$	generate next $Z$
2	$Z \leftarrow Z_s$	load seed



## WORD-SERIAL IMPLEMENTATION

In the word-serial implementation we add seven (or six) rotations of the current random number  $Z$  one-by-one

$$\begin{aligned} R &= (E_{14} + E_8 + E_7 + E_5 + E_2 + E_1 + E_0) \bmod M \\ &= (E_{14} + E_8 + E_7 + E_5 + E_3 - E_0) \bmod M \end{aligned}$$

$$\mathcal{E} = \begin{bmatrix} E_{14} \\ E_8 \\ E_7 \\ E_5 \\ E_2 \\ E_1 \\ E_0 \end{bmatrix} = \begin{bmatrix} z_{16} z_{15} z_{14} \dots z_0 z_{30} \dots z_{24} z_{23} z_{22} z_{21} z_{20} z_{19} z_{18} z_{17} & 7 \\ z_{22} z_{21} z_{20} \dots z_{30} z_{29} z_{28} z_{27} z_{26} z_{25} z_{24} z_{23} & 6 \\ z_{23} z_{22} z_{21} \dots z_0 z_{30} z_{29} z_{28} z_{27} z_{26} z_{25} z_{24} & 5 \\ z_{25} z_{24} z_{23} \dots z_2 z_1 z_0 z_{30} z_{29} z_{28} z_{27} z_{26} & 4 \\ z_{28} z_{27} z_{26} \dots z_5 z_4 z_3 z_2 z_1 z_0 z_{30} z_{29} & 3 \\ z_{29} z_{28} z_{27} \dots z_6 z_5 z_4 z_3 z_2 z_1 z_0 z_{30} & 2 \\ z_{30} z_{29} z_{28} \dots z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0 & 1 \end{bmatrix}$$

$R = Z;$

for  $k = 2:7$

$R = (R + \mathcal{E}(k, :)) \bmod M$

end

## BIT-SERIAL IMPLEMENTATION

A number of bit-serial implementations is possible. Two most obvious are

### - temporal bit-by-bit

In this case we have ONLY ONE

1-bit adder, and all operations

are expressed in terms of single-bit addition.

The main attraction of this solution is its hardware simplicity.

However the time required to perform the generation of the next random number is approx:

$$t_s = 7 \times 31 \times t_{ck} = 217 t_{ck}$$

For a 100 MHz clock  $t_s = 22 \mu s$

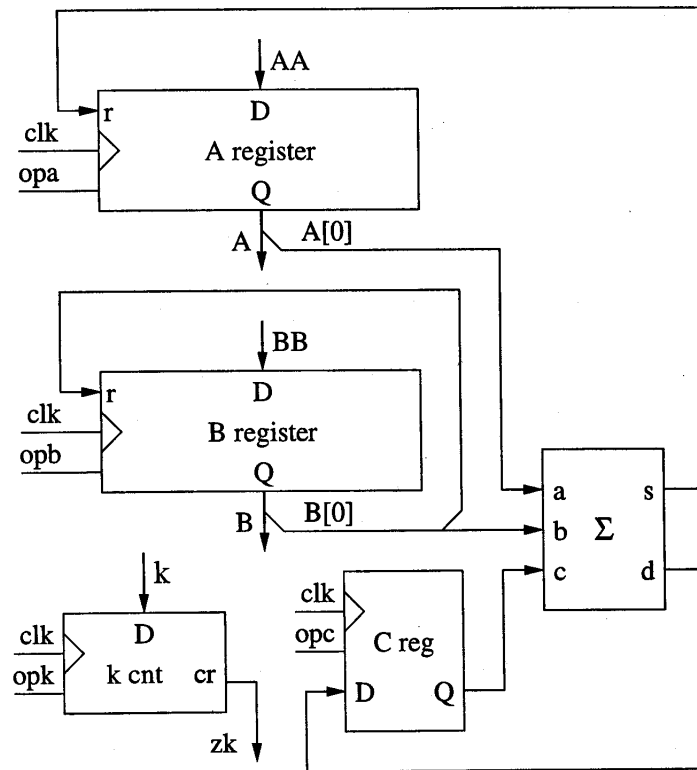


Figure 6-2: The datapath of the serial adder

As intended, we have three registers  $A$ ,  $B$  and  $C$ , the 1-bit adder  $\Sigma$ , and a step counter,  $k$ .

For each register, we have to specify the set of operations, typically some of operations like: hold (nop), load, shift, count, etc. Details have been discussed in sec. 5.5.

We are now ready to consider the **control unit**. A good way to start is to prepare a flow-chart of operations in a form of a **state transition diagram** which describes details of the algorithmic state machine.

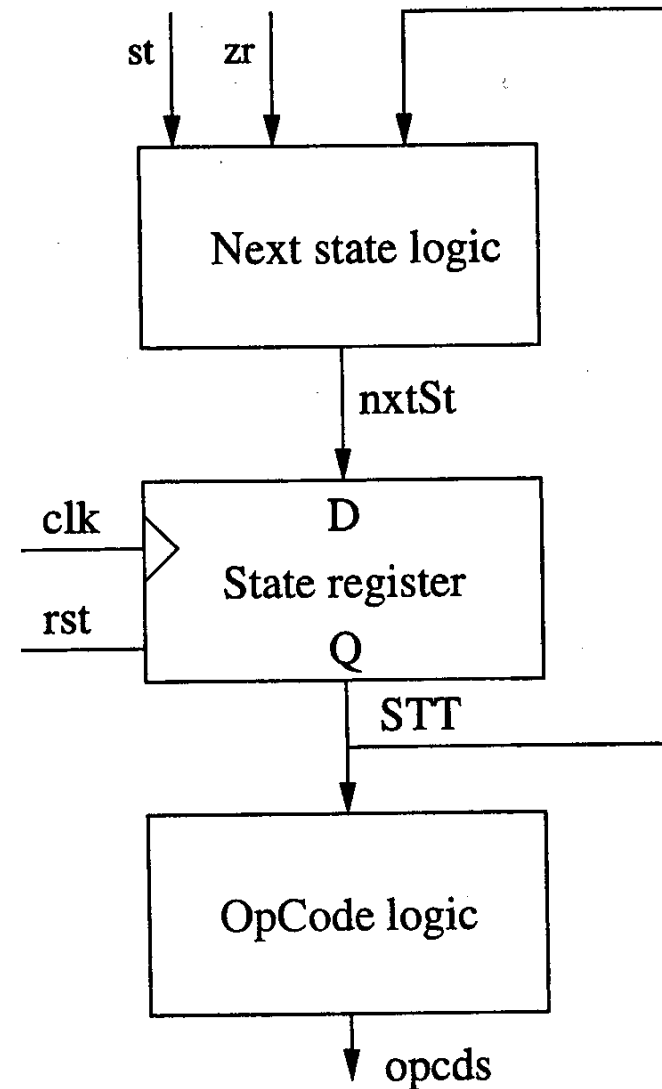


Figure 6-4: The control unit

### SPATIAL BIT-SERIAL IMPLEMENTATION

Consider again a problem of adding  
~~the~~ rotated  $Z$ 's as in the matrix  $E$

$$E = \begin{matrix} E_{14} \\ E_8 \\ E_7 \\ E_5 \\ E_2 \\ E_1 \\ E_0 \end{matrix} = \begin{matrix} z_{16} z_{15} z_{14} \dots z_0 z_{30} \dots z_{24} z_{23} z_{22} z_{21} z_{20} z_{19} z_{18} z_{17} \\ z_{22} z_{21} z_{20} \dots z_{30} z_{29} z_{28} z_{27} z_{26} z_{25} z_{24} z_{23} \\ z_{23} z_{22} z_{21} \dots z_0 z_{30} z_{29} z_{28} z_{27} z_{26} z_{25} z_{24} \\ z_{25} z_{24} z_{23} \dots z_2 z_1 z_0 z_{30} z_{29} z_{28} z_{27} z_{26} \\ z_{28} z_{27} z_{26} \dots z_5 z_4 z_3 z_2 z_1 z_0 z_{30} z_{29} \\ z_{29} z_{28} z_{27} \dots z_6 z_5 z_4 z_3 z_2 z_1 z_0 z_{30} \\ z_{30} z_{29} z_{28} \dots z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0 \end{matrix} \begin{matrix} g \\ f \\ e \\ d \\ c \\ b \\ a \end{matrix}$$

We can perform addition (mod  $M$ )  
 column-per-column, counting the  
 number of ones in each column.

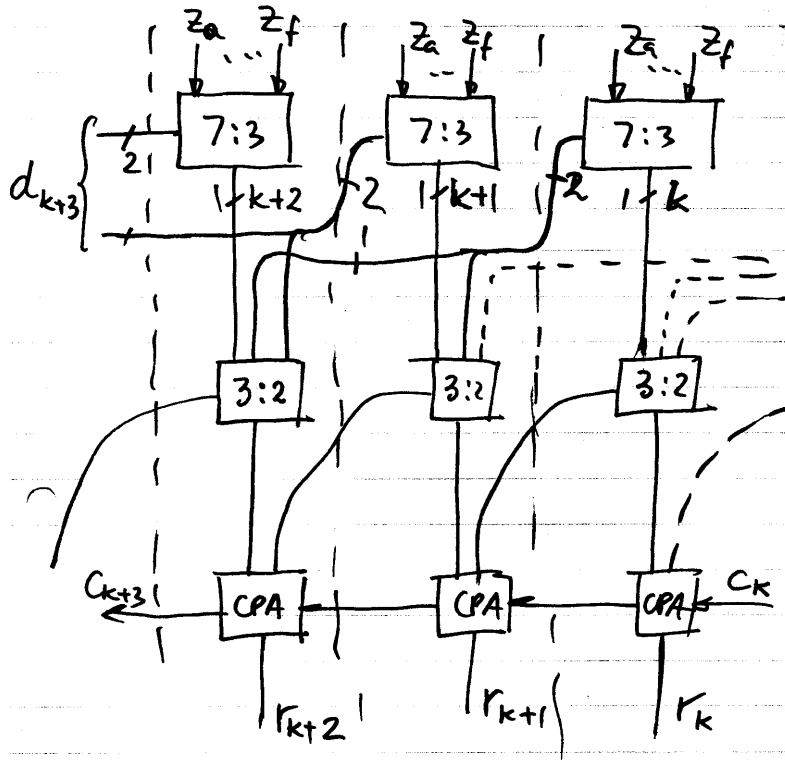
One way of doing so is as follows:

Adding seven bits we have

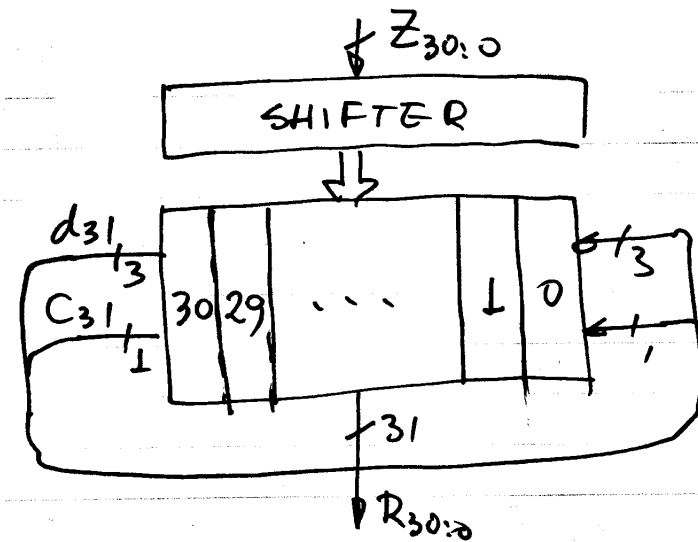
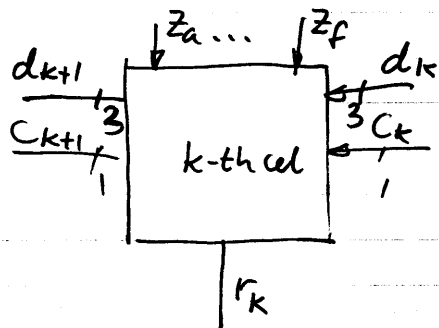
$$(z_a + z_b + z_c + z_d + z_e + z_f + z_g) = 2 \cdot d_{1:0} + S$$

Two-bit output carry  $d_{1:0}$  must now

be taken into account and we have:



The  $k$ -th cell consists of three-levels of adders: 7:3 CSA, 3:2 CSA and CPA





## References

- [1] A. P. Papliński and N. Bhattacharjee. Hardware implementation of the Lehmer random number generator. *IEE Proc.-Comput. Digit. Tech.*, 143(1):93–95, January 1996.
- [2] The MathWorks Inc. *Using MATLAB*, 2000.