# 3 Geração de números pseudo-aleatórios (pseudo_random_generator.vhd)

Diz-se que uma sequência de números $\ldots, x(-1), x(0), x(1), \ldots$, é pseudo-aleatória quando esta é gerada através de uma fórmula matemática, e quando a sequência assim gerada apresenta propriedades estatísticas parecidas com uma sequência de números aleatórios (daí o termo pseudo-aleatório).

## 3.1 A entidade pseudo_random_generator

A entidade pseudo_random_generator é capaz de fornecer um novo número pseudo-aleatório com, no máximo, 24 *bits* em cada ciclo de relógio. O seu *interface* é o seguinte:

```
entity pseudo_random_generator is
  generic
  (
    N_BITS : integer range 1 to 24;
    SEED   : std_logic_vector(47 downto 0)
  );
  port
  (
    clock  : in  std_logic;
    enable : in  std_logic := '1';
    rnd    : out std_logic_vector(N_BITS-1 downto 0)
  );
end pseudo_random_generator;
```

Descreve-se a seguir a função de cada um dos seus parâmetros/portos.

**N_BITS** Número de *bits* do número pseudo-aleatório a ser gerado.

**SEED** Semente do gerador. A semente é um número de 48 *bits*, por exemplo X"0123456789AB", que determina o estado inicial do gerador. Se for instanciado mais do que um gerador de números pseudo-aleatórios, é fortemente recomendado que sejam usadas ou sementes diferentes ou arquiteturas diferentes (existem duas).

**clock** Sinal de relógio.

**enable** Sinal de *enable*. O gerador só produz um novo número pseudo-aleatório se este sinal estiver a '1' (que é o valor por omissão se durante a instanciação da entidade não se disser nada acerca deste porto). Recomenda-se que se deixe o gerador sempre a trabalhar (enable sempre a '1'). Nesse caso, se os instantes nos quais o sinal rnd é usado pelo resto da lógica forem despoletados por ações externas à FPGA suficientemente espaçadas no tempo (por exemplo, ao se carregar num botão), então esse sinal será, para todos os efeitos práticos, verdadeiramente aleatório.

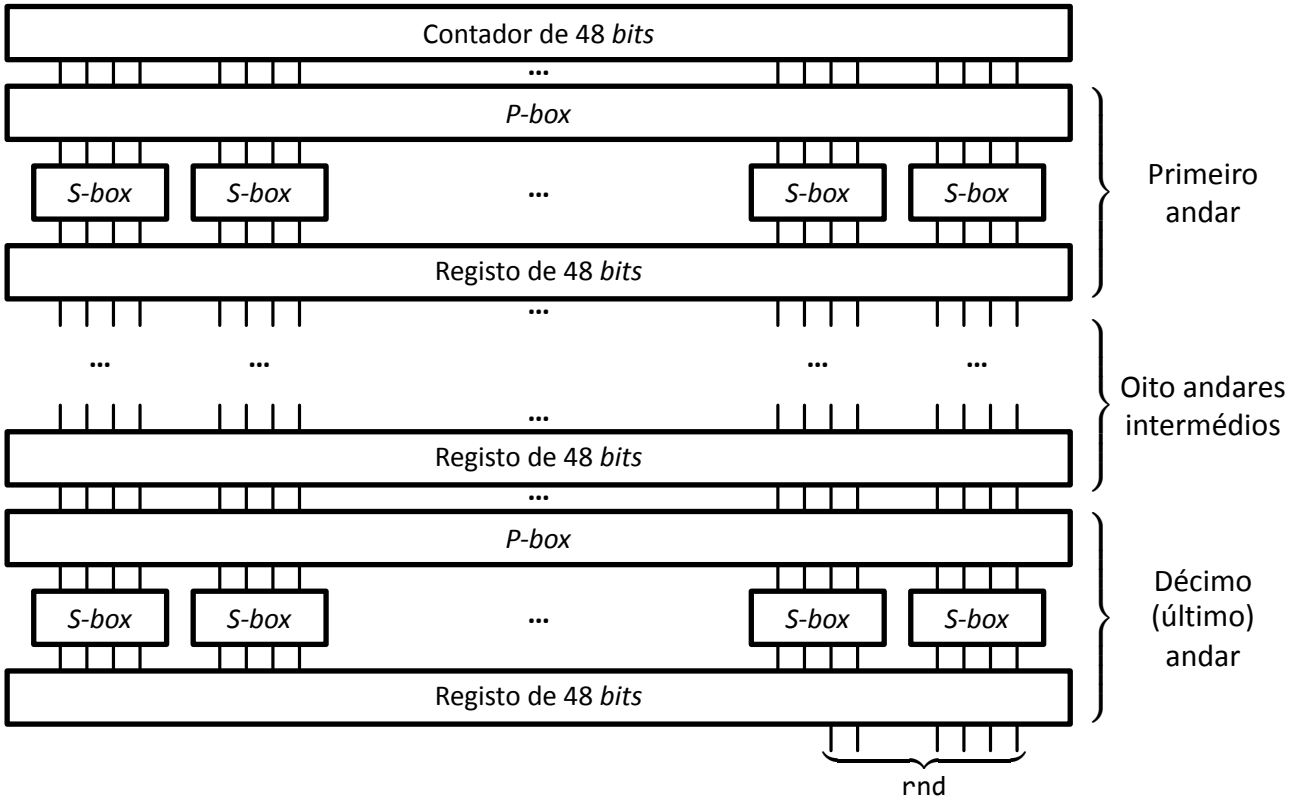
**rnd** Número pseudo-aleatório uniformemente distribuido gerado (calculado) pela entidade.

Para esta entidade existem duas arquiteturas, com nomes heavy e light, tendo o seu código VHDL sido gerado automaticamente por um programa escrito em C (pseudo_random_generator.c). A primeira usa uma quantidade de recursos da FPGA relativamente grande e foi desenhada tendo em conta algumas das boas práticas correntes relacionadas com segurança informática. A segunda é mais poupada em termos de recursos, mas produz números pseudo-aleatórios que são potencialmente de pior qualidade. Para reduzir ligeiramente os tempos de compilação, recomenda-se que seja utilizada a arquitetura light durante o desenvolvimento do projeto, e que se mude para a arquitetura heavy na sua versão final.

Esta entidade pode ser vista em ação no exemplo `pseudo_random_generator_example` e na demonstração `histogram`.

### 3.2 Princípio de funcionamento da arquitetura heavy

A figura seguinte mostra um diagrama de blocos do que está a acontecer dentro da arquitetura `heavy` (ilustrado para `N_BITS` igual a 6). Para não complicar a figura não se mostram as ligações dos sinais `clock` e `enable` ao contador e a todos os registos.




A ideia base por detrás do gerador consiste em baralhar os *bits* de um contador. O tipo de contador não é muito relevante (pode contar em binário, em código de Gray, ou outro); apenas é preciso que um contador de $n$ *bits* tenha $2^n$ estados diferentes. A baralhação dos *bits* do contador é feita em vários andares (quantos mais melhor) tal como ilustrado na figura, usando em cada um deles dois tipos de caixas:

> **caixas de permutação** (em Inglês *permutation boxes*, abreviadas para *P-box*), que baralham a ordem dos *bits* que por elas passam (por exemplo, o *bit* 0 da saída é o *bit* 12 da entrada, o *bit* 1 da saída é o *bit* 35 da entrada, *etc.*, sem nunca reutilizar um *bit* da entrada), e
>
> **caixas de substituição** (em Inglês *substitution boxes*, abreviadas para *S-box*), que no nosso caso transformam 4 *bits* em 4 *bits* (os números de 0 a 15 na entrada são transformados em números de 0 a 15 na saída, sem repetições).

As caixas de permutação devem ser diferentes umas das outras, o mesmo acontecendo com as caixas de substituição. Cada uma destas últimas deve ser escolhida de modo a que cada *bit* de saída seja influenciado por todos os *bits* de entrada. Deste modo cria-se um fenómeno de avalancha, que faz com que ao fim de vários andares (no nosso caso bastam 5) cada *bit* do registo na saída do andar seja influenciado por cada um dos *bits* do contador.

Cada andar do nosso gerador usa uma única *P-box* de 48 *bits* (uma *P-box* usa apenas recursos de encaminhamento na FPGA), e usa 12 *S-boxes* (cada uma delas usa 4 blocos de lógica configurável da FPGA). Excluindo o contador, para 10 andares são utilizadas no total 480 blocos de lógica configurável, sendo também usado em cada uma delas o *flip-flop* lá presente.

Deve-se usar como *bits* de saída do gerador no máximo metade dos *bits* presentes no seu andar final.

### 3.3 Princípio de funcionamento da arquitetura `light`

Nota para o leitor(a): talvez seja mais prudente avançar já para a secção seguinte. O material apresentado a seguir tem um teor matemático elevado cuja compreensão não é necessária para se fazer uma correta utilização da entidade `pseudo_random_generator`. Um bom aluno(a) deve ser capaz de compreender o que aqui é exposto sem grande dificuldade.

Note que apenas se descreve aqui uma das maneiras possíveis de gerar uma sequência de números inteiros pseudo-aleatória, maneira essa que é particularmente fácil de sintetizar numa FPGA.

#### 3.3.1 Aritmética modular

O nosso ponto de partida é a aritmética modular. Neste tipo de aritmética, depois de termos definido um módulo, que será um número inteiro positivo, apenas estamos interessados nos restos das divisões dos números inteiros por esse módulo. Por exemplo, quando o módulo é 12 e $k$ é um número inteiro, os números $\ldots, -10, 2, 14, 26, \ldots, 2 + 12k, \ldots$, são todos equivalentes, porque quando divididos por 12 dão todos resto 2. Para um módulo de $m$, teremos então $m$ classes de equivalência, cada uma delas correspondendo a um resto $r$, com $0 \leqslant r < m$. É usual representar uma classe de equivalência apenas pelo seu resto $r$.

As operações aritméticas de soma, subtração e multiplicação são feitas da maneira habitual (tendo em atenção que estamos apenas interessados no resto). Apresentamos de seguida as tabelas das operações soma e multiplicação para os módulos 2 e 3.

| $+$ | $0$ | $1$ |
|---|---|---|
| $0$ | $0$ | $1$ |
| $1$ | $1$ | $0$ |

| $\times$ | $0$ | $1$ |
|---|---|---|
| $0$ | $0$ | $0$ |
| $1$ | $0$ | $1$ |

| $+$ | $0$ | $1$ | $2$ |
|---|---|---|---|
| $0$ | $0$ | $1$ | $2$ |
| $1$ | $1$ | $2$ | $0$ |
| $2$ | $2$ | $0$ | $1$ |

| $\times$ | $0$ | $1$ | $2$ |
|---|---|---|---|
| $0$ | $0$ | $0$ | $0$ |
| $1$ | $0$ | $1$ | $2$ |
| $2$ | $0$ | $2$ | $1$ |

Note que a operação de adição módulo 2 corresponde à operação lógica ou-exclusivo (`xor`) e que a de multiplicação módulo 2 corresponde à operação lógica e (`and`), pelo que trabalhar módulo $2$ é particularmente fácil.

Para um módulo genérico $m$, o conjunto das classes de equivalência e as operações $+$ e $\times$ definem um anel. Em geral, nem todas as classes de equivalência possuem inverso; isso apenas acontece quando o máximo divisor comum entre $r$ e $m$ é $1$. Como a operação de divisão é importante em algumas aplicações, é usuar restringir o módulo a ser um número primo $p$, porque neste caso apenas o elemento neutro da adição, $r = 0$, não tem inverso. Neste caso o anel é também um corpo (é um caso particular de um corpo finito), usualmente designado por $\mathbb{F}_p$. Grosso modo, afirmar que algo pertence a $\mathbb{F}_p$ diz-nos que esse algo é um número inteiro maior ou igual a $0$ e menor do que $p$, e que as operações aritméticas efetuados com esse inteiro são feitas com aritmética modular (com módulo $p$).

Daqui para diante o módulo será sempre um número primo $p$. Na prática, será o número $2$, mas a explicação da teoria pode, e deve, ser feita para o caso mais geral.

#### 3.3.2 Fórmula matemática que vamos utilizar

A fórmula matemática que vamos usar para gerar números pseudo-aleatórios é extremamente simples: dado o estado atual do gerador, o estado seguinte é calculado através de uma simples multiplicação. Como estamos a trabalhar com aritmética modular, o número de estados possíveis do gerador é finito, pelo que a sequência que vai ser gerada entrará, mais tarde ou mais cedo, num ciclo. Para permitir sequências com ciclos com um período longo quando se usa um módulo $p$ pequeno, o estado do nosso gerador no instante $t$

será um conjunto de $n$ números pertencentes a $\mathbb{F}_p$, que vamos designar por $x_{n-1}(t), \ldots, x_0(t)$ e que vamos agrupar no vetor coluna $u(t)$:

$$u(t) = \begin{bmatrix} x_{n-1}(t) \\ x_{n-2}(t) \\ \vdots \\ x_1(t) \\ x_0(t) \end{bmatrix}.$$

A nossa fórmula para atualizar o estado do gerador será

$$u(t+1) = Au(t),$$

onde $A$ é uma matriz quadrada invertível (não singular) de dimensões $n \times n$. Como estamos a trabalhar com aritmética modular, cada entrada da matriz e o seu determinante também pertencem a $\mathbb{F}_p$ (o determinante não pode ser $0$ porque não queremos matrizes singulares). Como consequência desta fórmula é possível calcular rapidamente o estado do gerador no instante de tempo $t_2$ a partir do seu estado no instante $t_1$:

$$u(t_2) = A^{t_2 - t_1} u(t_1).$$

Como a matriz $A$ é invertível, não é preciso que $t_2$ seja maior do que $t_1$; é perfeitamente possível andar para trás no tempo. (Por este, e por outros motivos que não vamos expôr aqui, a utilização de um gerador de números pseudo-aleatórios deste tipo é altamente desaconselhada em aplicações que exigem elevado sigilo.)

Cada um dos estados possíveis (são $p^n$ ao todo) faz parte de um ciclo; estados diferentes podem, é claro, fazer parte de ciclos diferentes. Mais uma vez, isso é uma consequência do facto de $A$ ser uma matriz invertível. Em particular, o estado zero (tudo a zero) dá sempre origem a um ciclo de período $1$ (ponto fixo). Este caso deve ser evitado a todo o custo, já que a sequência resultante, sempre zero, não é aleatória. Felizmente este caso pode ser evitado inicializando o gerador de números pseudo-aleatórios com um estado inicial diferente de tudo a zero.

Restam portanto $p^n - 1$ estados. Será que é possível escolher a matriz $A$ de modo a que todos esses estados formem um único ciclo, de período $p^n - 1$? A resposta é sim. Mais ainda, existem muitas matrizes com essa propriedade pelo que encontrar uma é uma tarefa relativamente simples. Atendendo a que $u(t) = A^t u(0)$, para que o período seja $p^n - 1$ é necessário que $A^{p^n-1}$ seja a matriz identidade. Para que o período não seja mais pequeno é necessário que $A^{(p^n-1)/f}$ não seja a matriz identidade, sendo $f$ um factor de $p^n - 1$. Para testar esta última condição é suficiente restringir $f$ aos números primos que são factores de $p^n - 1$.

Na arquitetura `light` da entidade `pseudo_random_generator` foi usada uma matriz de dimensões $48 \times 48$, escolhida aleatoriamente de modo a satisfazer as condições descritas no parágrafo anterior e sujeitas às restrições de que i) o número de uns em cada linha (o *fan-in*) da matriz não é nem inferior a 2 nem superior a 4, e ii) o número de uns de cada coluna (o *fan-out*) não é nem inferior a 2 nem superior a 5. O período do gerador é $2^{48} - 1 \approx 2.815 \times 10^{14}$. Como são disponibilizados no máximo apenas 24 *bits* para o exterior, parte do estado do gerador não é diretamente exposto. O parâmetro genérico SEED define o estado inicial $u(0)$ do gerador. Note que foram tomadas precauções para garantir que o gerador não fica preso no estado tudo a zero.

### 3.3.3 Um caso particular muito usado na prática

Apesar de ser possível gerar uma sequência pseudo-aleatória de período máximo usando uma matriz $A$ sem nenhuma estrutura especial, isto é, com a maioria dos seus $n^2$ elementos diferentes de zero, do ponto de vista prático é muitas vezes preferível usar uma matriz $A$ em que a grande maioria dos seus elementos

é zero. Em particular, é possível escolher $n$ elementos de $\mathbb{F}_p$, que vamos designar por $a_0, \ldots, a_{n-1}$, de modo a que a matriz

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_2 & a_1 & a_0 \end{bmatrix}$$

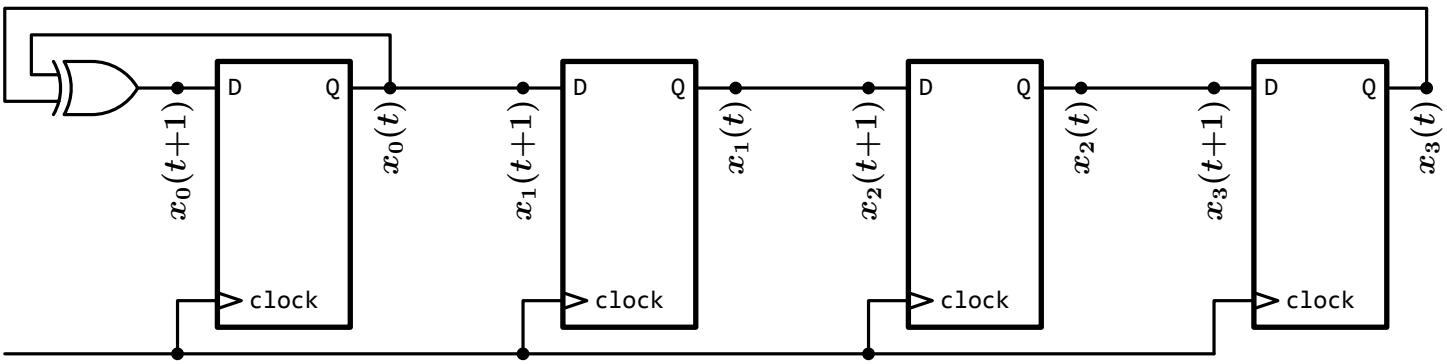
satisfaça as condições de período máximo. A uma matriz com esta forma dá-se o nome de matriz companheira. No nosso caso, como a matriz não pode ser singular, teremos necessariamente de ter $a_{n-1} \neq 0$. Com a matriz $A$ nesta forma a atualização do estado do gerador é feita usando a fórmula

$$\begin{bmatrix} x_{n-1}(t+1) \\ x_{n-2}(t+1) \\ x_{n-3}(t+1) \\ \vdots \\ x_2(t+1) \\ x_1(t+1) \\ x_0(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} x_{n-1}(t) \\ x_{n-2}(t) \\ x_{n-3}(t) \\ \vdots \\ x_2(t) \\ x_1(t) \\ x_0(t) \end{bmatrix},$$

ou seja, temos

$$\begin{aligned} x_{n-1}(t+1) &= x_{n-2}(t) \\ x_{n-2}(t+1) &= x_{n-3}(t) \\ &\vdots \\ x_2(t+1) &= x_1(t) \\ x_1(t+1) &= x_0(t) \\ x_0(t+1) &= a_{n-1}x_{n-1}(t) + a_{n-2}x_{n-2}(t) + \cdots + a_0 x_0(t) = \textstyle\sum_{k=0}^{n-1} a_k x_k(t). \end{aligned}$$

No caso $p = 2$ cada um dos $x_k(t)$ é um único *bit* pelo que as equações apresentadas acima descrevem o funcionamento de um registo de deslocamento (*shift register* em Inglês), no qual o *bit* que entra no registo de deslocamento é o ou-exclusivo de alguns dos *bits* internos do estado anterior do mesmo registo (trata-se de um *Linear Feedback Shift Register*). Em particular, na fórmula $x_0(t+1) = \sum_{k=0}^{n-1} a_k x_k(t)$ podemos eliminar as parcelas em que $a_k = 0$ e podemos eliminar as multiplicações por um nas em que $a_k = 1$. Tudo isto faz com que a geração de *bits* pseudo-aleatórios por este método seja muito eficiente e use muito poucos recursos (quer em lógica combinacional quer em registos), como se ilustra na figura seguinte para $n = 4$, $a_0 = 1$, $a_1 = 0$, $a_2 = 0$ e $a_3 = 1$ (pelo que $x_0(t+1) = x_0(t) + x_3(t)$; lembre-se que uma soma módulo 2 é um ou-exclusivo).



Neste caso concreto o gerador tem dois ciclos, um de período 1 no qual o estado é sempre 0000, caso que queremos evitar, e o outro de período 15 no qual o estado do gerador percorre sequencialmente todos os estádios do ciclo 0001, 0011, 0111, 1111, 1110, 1101, 1010, 0101, 1011, 0110, 1100, 1001, 0010, 0100, 1000. Neste último caso, e olhando para apenas um dos *bits* do gerador (não importa qual) verifica-se que num ciclo este toma o valor 0 sete vezes e o valor 1 oito vezes. Logo, os *bits* pseudo-aleatórios gerados não estão distribuidos uniformemente. Para valores elevados de $n$ a diferença entre os dois casos ($2^{n-1}-1$ contra $2^{n-1}$) é irrisória.

Esta maneira de gerar *bits* pseudo-aleatórios, com a matriz $A$ escolhida como descrito no slide anterior, tem, no que diz respeito à sua implementação, a desvantagem de que a lógica combinacional (os ou-exclusivos) está toda concentrada no cálculo de $x_0(t+1)$, o que pode levar a que a frequência máxima de funcionamento do gerador não seja a mais alta possível (veja adiante como esse problema pode ser resolvido), o que pode acontecer se o número de coeficientes $a_k$ diferentes de zero for maior do que dois. Felizmente isto não é um grande problema, porque para bastantes valores de $n$ é possível ter apenas dois dos $a_k$ iguais a um, e para todos os valores de $n$ maiores do que 4 é possível ter apenas quatro dos $a_k$ iguais a um.

Note que nos livros de engenharia sobre este assunto é usual apresentar este caso particular noutros moldes, usando o conceito de polinómio primitivo. Na opinião do autor deste documento é muito mais elegante e simples apresentar a teoria da maneira como foi aqui feita, que tem como pano de fundo o grupo linear geral das matrizes não singulares (*general linear group* em Inglês).

### 3.3.4 Outro caso particular muito usado na prática

Além da forma especial da matriz $A$ descrita anteriormente, a forma seguinte, que é uma matriz companheira orientada de outra maneira e é igualmente útil, é por vezes utilizada (para distinguir as duas formas, neste segundo caso vamos passar a usar a letra B em vez da letra A):

$$B = \begin{bmatrix} -b_{n-1} & 1 & 0 & \cdots & 0 & 0 & 0 \\ -b_{n-2} & 0 & 1 & \cdots & 0 & 0 & 0 \\ -b_{n-3} & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -b_2 & 0 & 0 & \cdots & 0 & 1 & 0 \\ -b_1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ -b_0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix}.$$

Neste caso temos de ter $b_0 \neq 0$, e as fórmulas de atualização do estado do gerador são

$$\begin{bmatrix} x_{n-1}(t+1) \\ x_{n-2}(t+1) \\ x_{n-3}(t+1) \\ \vdots \\ x_2(t+1) \\ x_1(t+1) \\ x_0(t+1) \end{bmatrix} = \begin{bmatrix} -b_{n-1} & 1 & 0 & \cdots & 0 & 0 & 0 \\ -b_{n-2} & 0 & 1 & \cdots & 0 & 0 & 0 \\ -b_{n-3} & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -b_2 & 0 & 0 & \cdots & 0 & 1 & 0 \\ -b_1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ -b_0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{n-1}(t) \\ x_{n-2}(t) \\ x_{n-3}(t) \\ \vdots \\ x_2(t) \\ x_1(t) \\ x_0(t) \end{bmatrix},$$

ou seja,

$$x_k(t+1) = \begin{cases} x_{k-1}(t) - b_k x_{n-1}(t), & \text{para } k = 1, 2, \ldots, n-1, \\ -b_0 x_{n-1}(t), & \text{para } k = 0. \end{cases}$$

Neste caso a lógica combinacional está espalhada ao longo do registo de deslocamento, em vez de estar toda concentrada no cálculo de $x_0(t+1)$. É pois preferível usar esta forma quando se pretender atingir uma frequência de funcionamento do circuito o mais elevada possível.

É interessante constatar que se o estado do gerador for representado pelo polinómio (na variável $X$)

$$P(t;X) = \sum_{k=0}^{n-1} x_k(t) X^k,$$

definindo

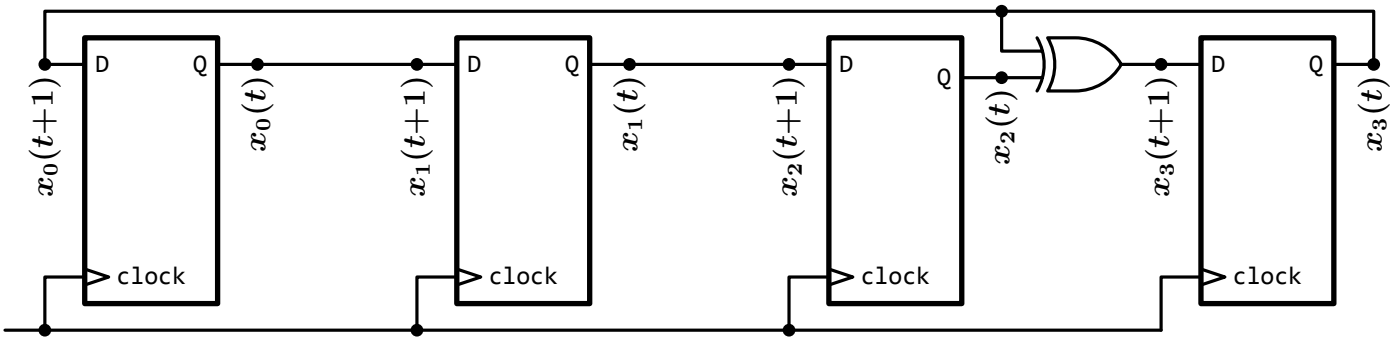
$$B(X) = X^n + \sum_{k=0}^{n-1} b_k X^k$$

(trata-se do tal polinómio primitivo a que se fez alusão atrás), e atendendo a que

$$XP(t,X) = \sum_{k=0}^{n-1} x_k(t) X^{k+1} = x_{n-1} X^n + \cdots,$$

verifica-se que o resto da divisão de $XP(t,X)$ por $B(x)$ é dado por $XP(t;X) - x_{n-1}(t)B(X)$. Ora é isso mesmo que a fórmula apresentada no fim do parágrafo anterior faz, pelo que $P(t+1;X)$ é o resto da divisão de $XP(t;X)$ por $B(X)$, sendo obviamente as operações aritméticas feitas em $\mathbb{F}_p$. É pois possível fazer divisões de polinómios com registos de deslocamento (em base 2 isto é particularmente fácil).

Para $p = 2$, $n = 4$, $b_0 = 1$, $b_1 = 0$, $b_2 = 0$ e $b_3 = 1$, o diagrama de blocos do gerador toma a forma da figura seguinte (neste caso $-0 = 0$ e $-1 = 1$, pelo que podemos substituir todas as subtrações por somas).




Note que substituindo $x_0(t+1) = x_3(t)$ por $x_0(t+1) = x_3(t) + y(t)$ o circuito da figura anterior pode ser usado para calcular um *cyclic redundancy checksum* (CRC) de $4$ *bits* da sequência $y(t)$.