Cryptography Part XI: Use of Register-based Structures in Cryptography

Cryptographic Hardware for Embedded Systems

ECE 3170

Fall 2025

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Reading

- Please read chapter 17.4 and 17.6 of the course textbook by Schneier
- Please also note that these lecture notes contain significant additional content not in Schneier

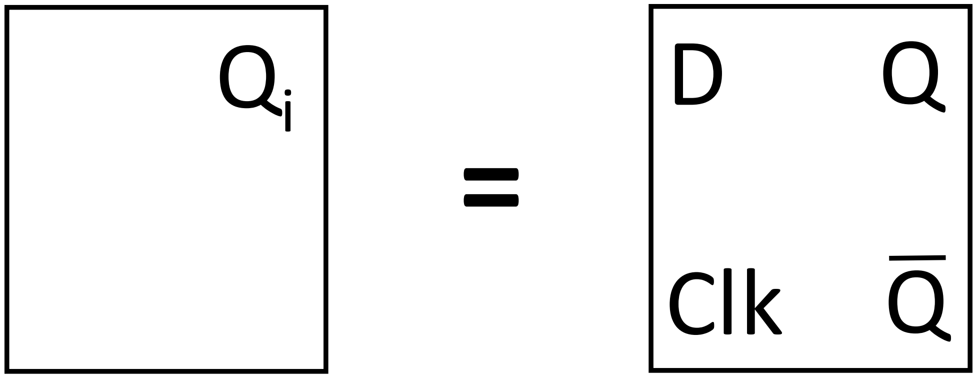
Math Background: Definitions

- A *group* (G,*) consists of a set G with a binary operation * on G satisfying the following three axioms:
  i. The group operation is *associative*: a*(b*c) = (a*b)*c for all a, b, c ϵ G
  ii. There is an element 1 ϵ G, called the *identity element*, such that a*1=1*a=a for all a ϵ G
  iii. For each a ϵ G there exists an element a^{-1} ϵ G, called the inverse of a, such that a*a=a*a=1
- A group G is Abelian (or commutative) if, furthermore,
  iv. a*b = b*a for all a, b ϵ G

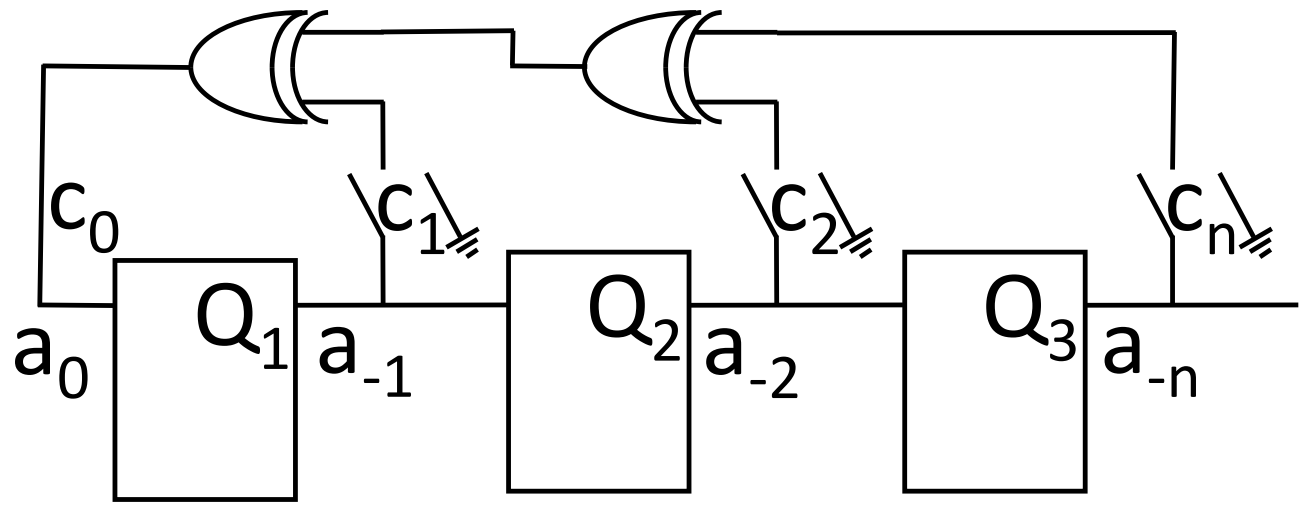
Math Background: Definitions continued

- A *ring* $(R,+,\times)$ consists of a set R with two binary operations arbitrarily denoted + (addition) and × (multiplication) on R, satisfying the following axioms:
  1. $(R,+)$ is an Abelian group with identity denoted 0.
  2. The operation × is associative: $a \times (b \times c) = (a \times b) \times c$ for all $a, b, c \in R$
  3. There is a multiplicative identity denoted 1, with $1 \neq 0$, such that $1 \times a = a \times 1 = a$ for all $a \in R$
  4. The operation × is *distributive* over +. That is, $a \times (b + c) = (a \times b) + (a \times c)$ for all $a,b,c \in R$
- The ring is a *commutative ring* if $a \times b = b \times a$ for all $a, b \in R$
- A *field* is a commutative ring in which all non-zero elements have multiplicative inverses

Notation

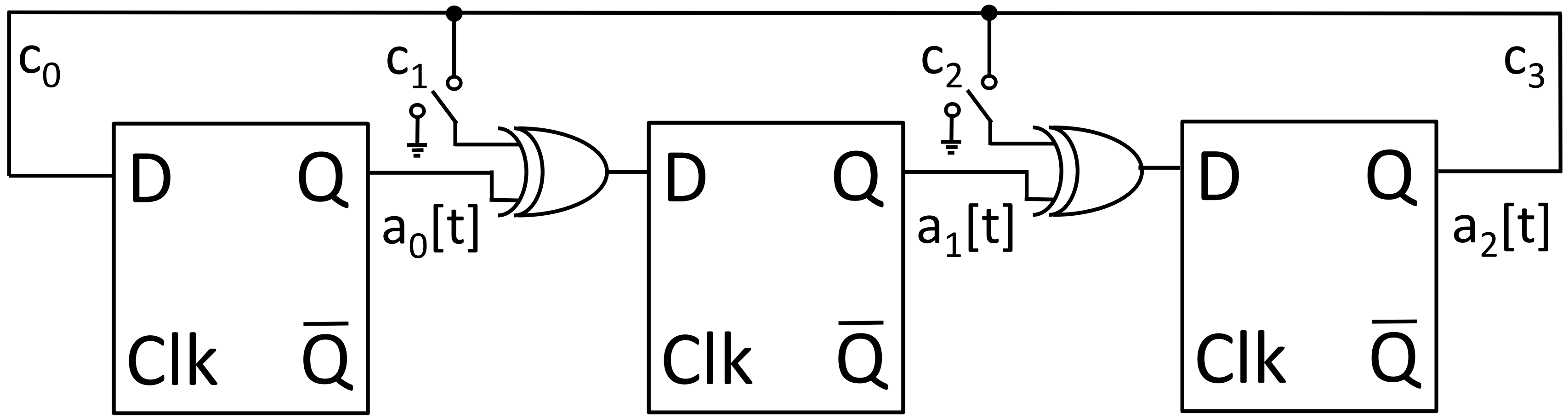


LFSR Notation 1



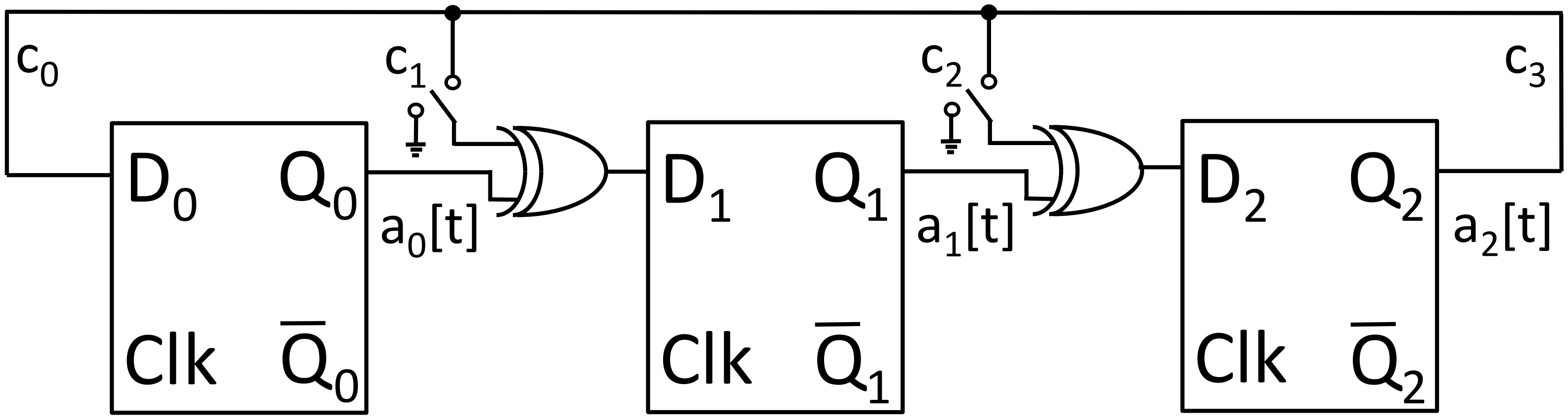
- Some authors call this type of LFSR “external-XOR”; others “Fibonacci”
- For n FFs, the initial state is a_0, a_{-1}, a_{-2}, …, a_{-n}
- Note that we always have c_0 = 1 (feedin to register Q_1) and c_n = 1, i.e., the switch for the last tap makes a feedback connection
  - (c_n = 0 would imply connection to ground)

LFSR Notation 2



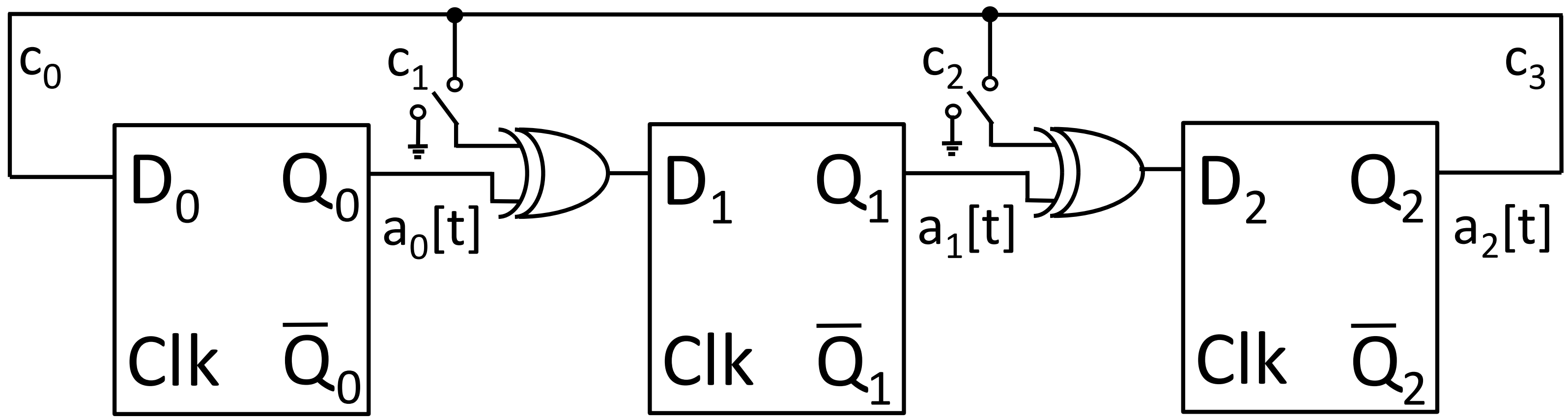
- Some authors call this type of LFSR “internal-XOR”; others “Galois”
- For n FFs, the initial state is $a_0[0]$, $a_1[0]$, $a_2[0]$, …, $a_{n-1}[0]$
- Note that we always have $c_0 = 1$ (feedin to the first register) and $c_n = 1$, i.e., the output for the last register is always fed back

Bit vectors as field elements



Example 1. Let the state of the *n*-bit Linear Feedback Shift Register (LFSR) above be expressed as *A(x)[t]* = $\langle a_{n-1}[t], \ldots, a_2[t], a_1[t], a_0[t]\rangle$ which can be represented in polynomial space as $a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x + a_0$ where x^i, $0 \le i \le n-1$, is indeterminate, i.e., x^i is not evaluated numerically but indicates position – specifically, a_i is the output of flip-flop Q_i and thus a_i is the coefficient associated with x^i in the polynomial. Polynomial addition and multiplication are formed with coefficients added modulo 2, i.e., with exclusive-OR logic. Each clock causes this LFSR to perform polynomial modulo on *A(x)[t]* shifted by one flip flop (i.e., multiplied by x) with the modulo polynomial $c_nx^n + c_{n-1}x^{n-1} + \cdots + c_2x^2 + c_1x + c_0$ which is also known as the characteristic polynomial *P(x)* of the LFSR.

Bit vectors as field elements (cont'd)

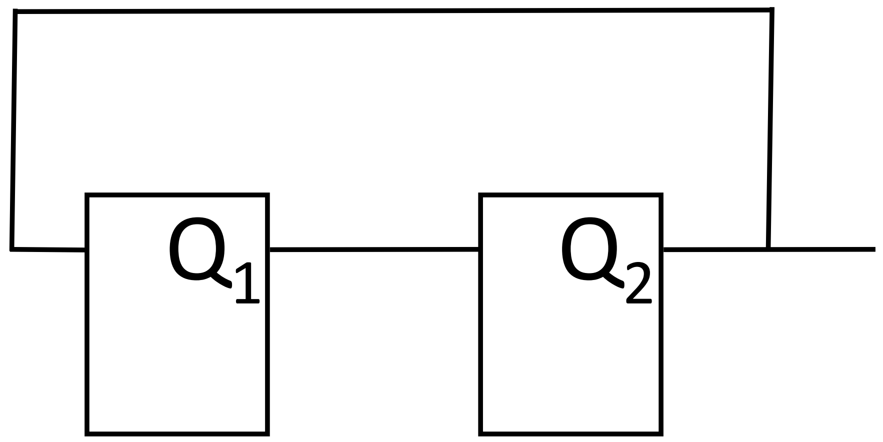


A(x)[t] = $\langle a_{n-1}[t], \dots, a_2[t], a_1[t], a_0[t] \rangle$ which can be represented in polynomial space as $\sum_{i=0}^{n-1} a_i x^i = a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$

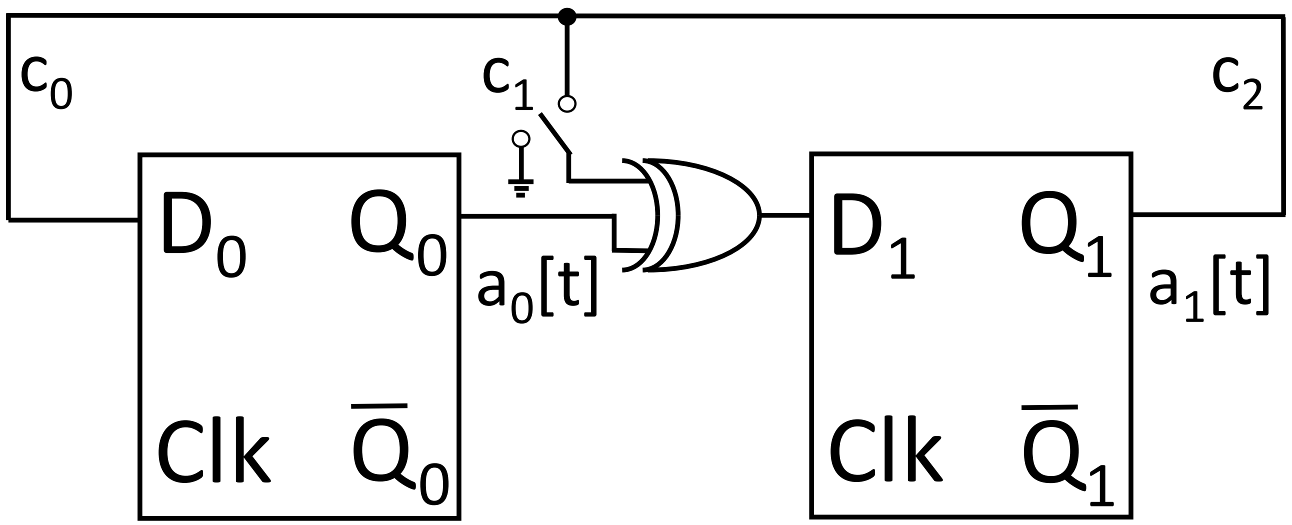
A(x)[t+1] = $\langle a_{n-1}[t+1], \dots, a_2[t+1], a_1[t+1], a_0[t+1] \rangle$ = *xA(x)[t]* mod *P(x)*

$P(x) = \sum_{i=0}^{n} c_i x^i = x^n + c_{n-1}x^{n-1} + \cdots + c_2 x^2 + c_1 x + 1$ since we always have c_n = 1 and c_0 = 1

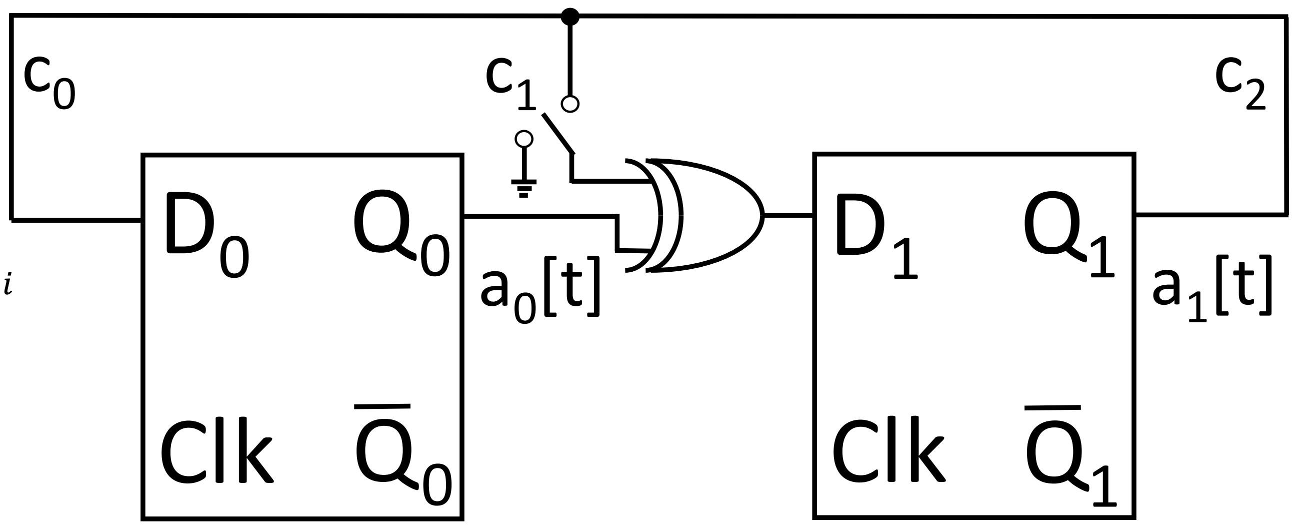
Example 1



Example 2

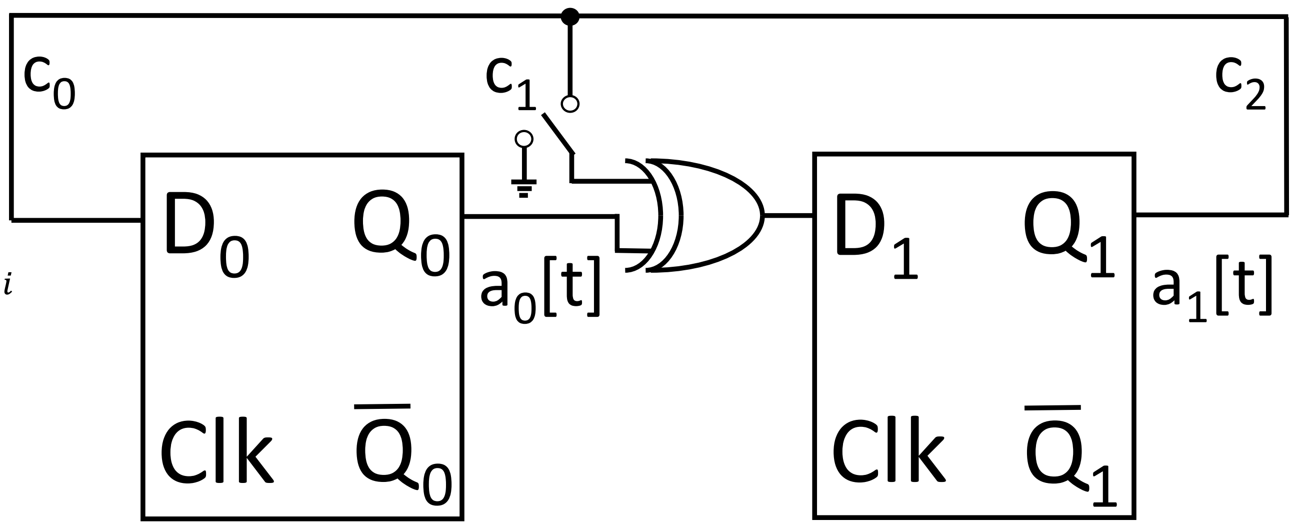


Example 2



A(x)[t] = $\langle a_1[t], a_0[t]\rangle$ = $\sum_{i=0}^{n-1} a_i[t]x^i$
$= a_1[t]x + a_0[t]$

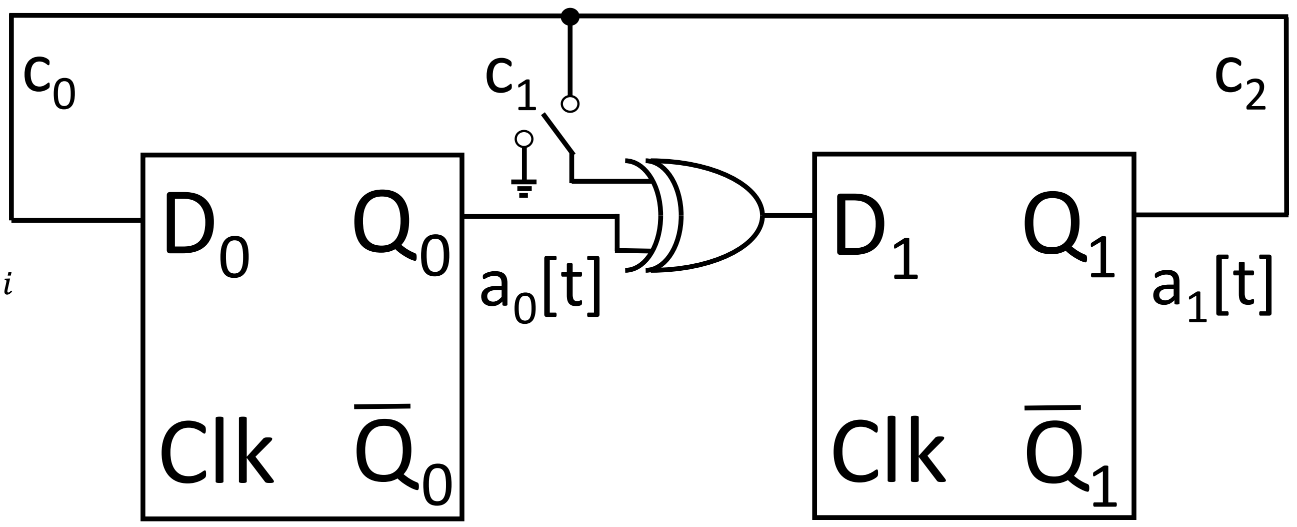
Example 2



$A(x)[t] = \langle a_1[t], a_0[t]\rangle = \sum_{i=0}^{n-1} a_i[t]x^i$
$= a_1[t]x + a_0[t]$

$P(x) = \sum_{i=0}^{n} c_i x^i = x^2 + c_1 x + 1$

Example 2

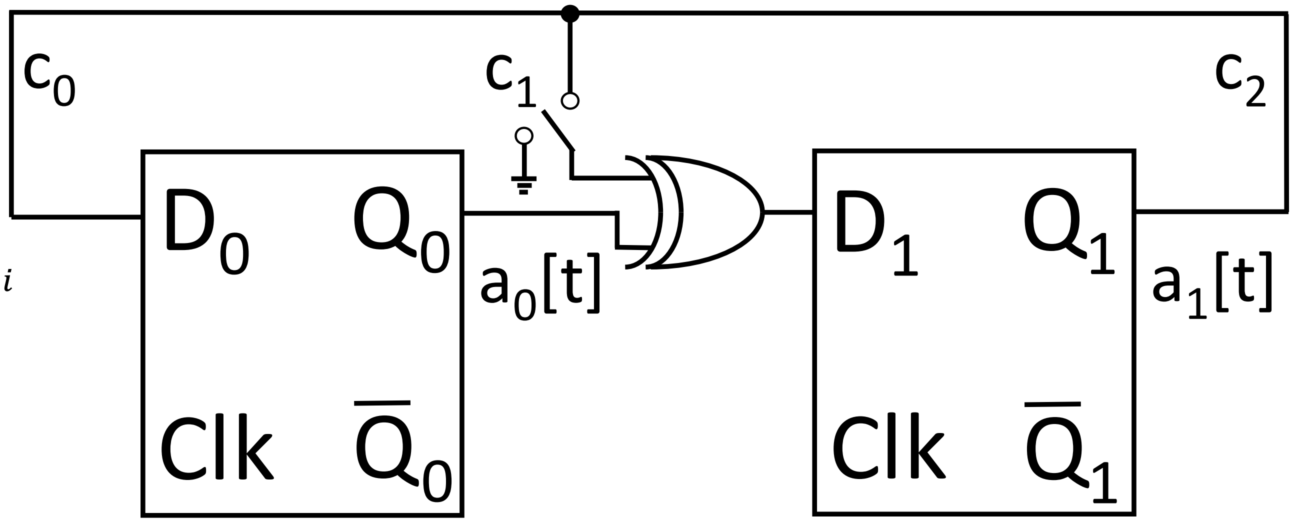


A(x)[t] = $\langle a_1[t], a_0[t]\rangle$ = $\sum_{i=0}^{n-1} a_i[t]x^i$
$= a_1[t]x + a_0[t]$

$P(x) = \sum_{i=0}^{n} c_i x^i$ = $x^2 + c_1 x + 1$

A(x)[t+1] = $\langle a_1[t+1], a_0[t+1]\rangle$ = *xA(x)[t]* mod *P(x)*

Example 2



A(x)[t] = $\langle a_1[t], a_0[t]\rangle$ = $\sum_{i=0}^{n-1} a_i[t]x^i$
$= a_1[t]x + a_0[t]$

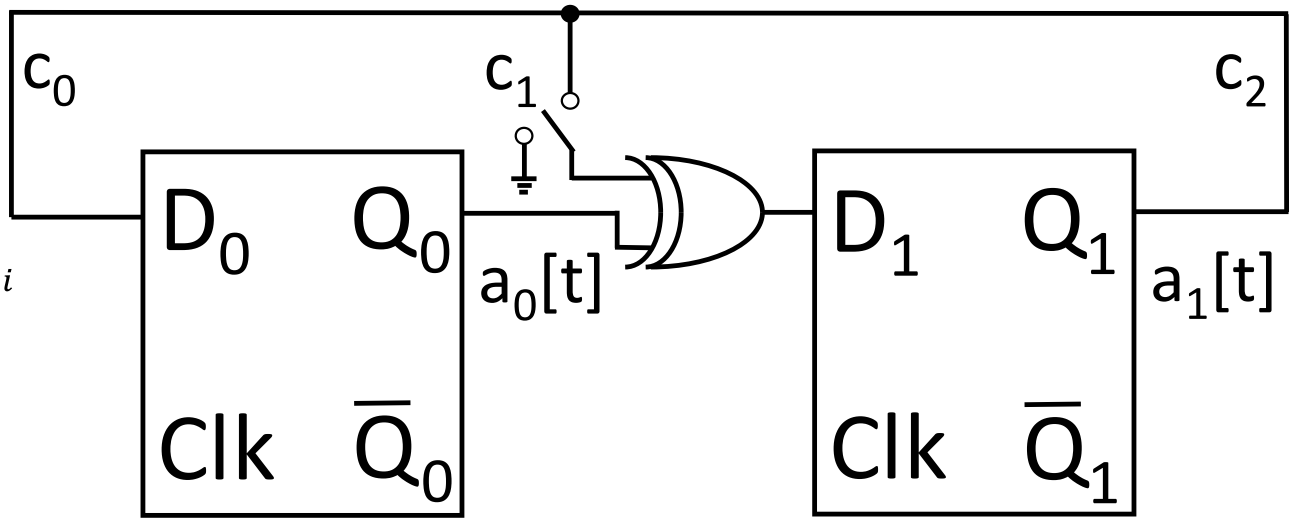
$P(x) = \sum_{i=0}^{n} c_i x^i$ = $x^2 + c_1 x + 1$

A(x)[t+1] = $\langle a_1[t+1], a_0[t+1]\rangle$ = *xA(x)[t]* mod *P(x)*

A(x)[t+1] = $(x(a_1[t]x + a_0[t]))$ mod x^2+x+1

A(x)[t+1] = $(a_1[t]x^2 + a_0[t]x)$ mod x^2+x+1

Example 2



$A(x)[t] = \langle a_1[t], a_0[t]\rangle = \sum_{i=0}^{n-1} a_i[t]x^i$
$= a_1[t]x + a_0[t]$

$P(x) = \sum_{i=0}^{n} c_i x^i = x^2 + c_1 x + 1$

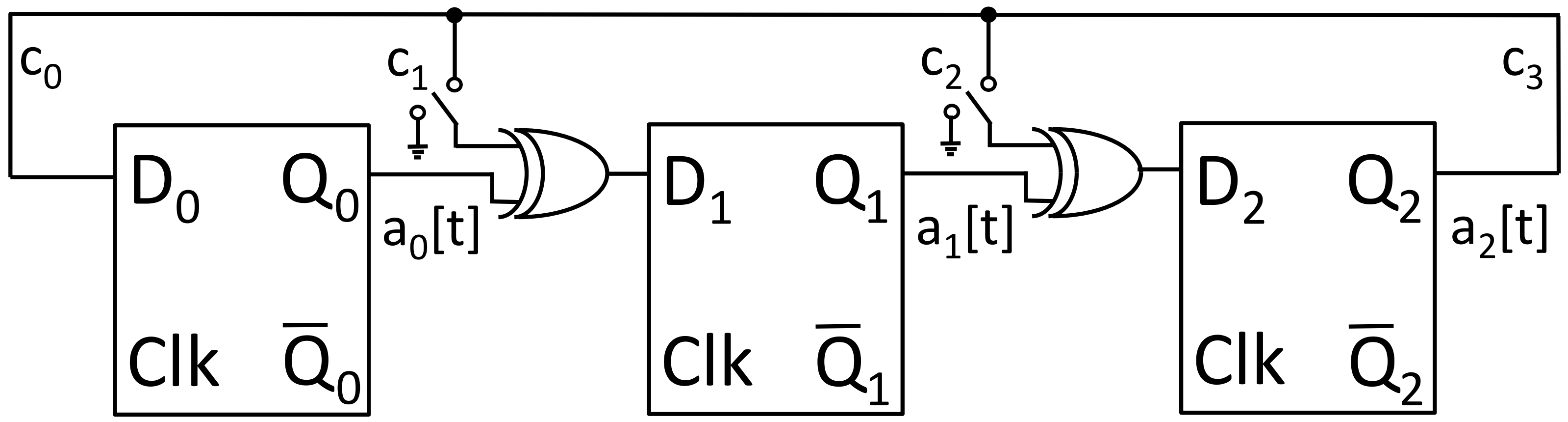
$A(x)[t+1] = \langle a_1[t+1], a_0[t+1]\rangle = xA(x)[t] \bmod P(x)$

$A(x)[t+1] = (x(a_1[t]x + a_0[t])) \bmod x^2+x+1$

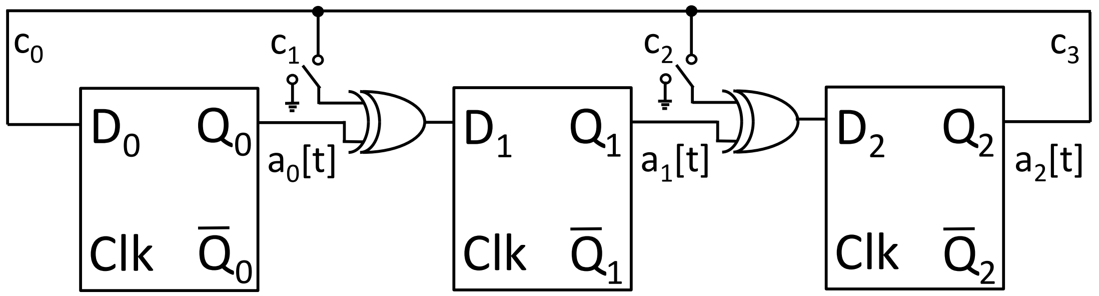
$A(x)[t+1] = (a_1[t]x^2 + a_0[t]x) \bmod x^2+x+1$

$A(x)[t+1] = (a_1[t] + a_0[t])x + a_1[t] = \langle a_1[t] + a_0[t], a_1[t]\rangle = \langle a_1[t+1], a_0[t+1]\rangle$

Example 3


c_0
c_1
c_2
c_3
D_0
Q_0
Clk
$\overline{Q}_0$
$a_0[t]$
D_1
Q_1
Clk
$\overline{Q}_1$
$a_1[t]$
D_2
Q_2
Clk
$\overline{Q}_2$
$a_2[t]$

Linear Feedback Shift Register (LFSR) Example 4



=

Example 2: Consider an LFSR (with the notation as defined) with n = 3, c_2=1 and c_1=0.

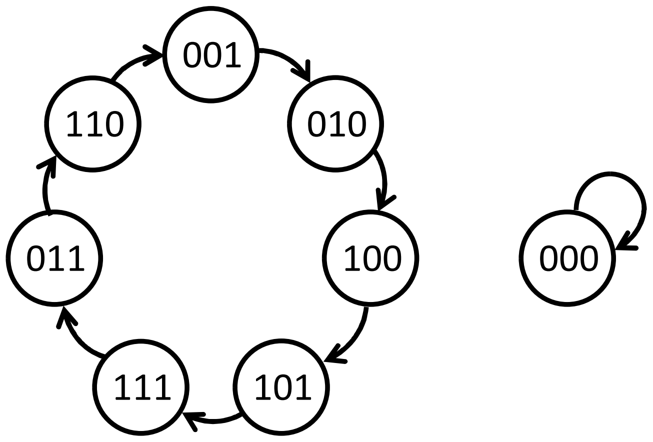
LFSR:
$P(x) = x^3 + x^2 + 1$
Update implemented by flip flops = x

Input:	000	001	010	011	100	101	110	111
Output:	000	010	100	110	101	111	001	011

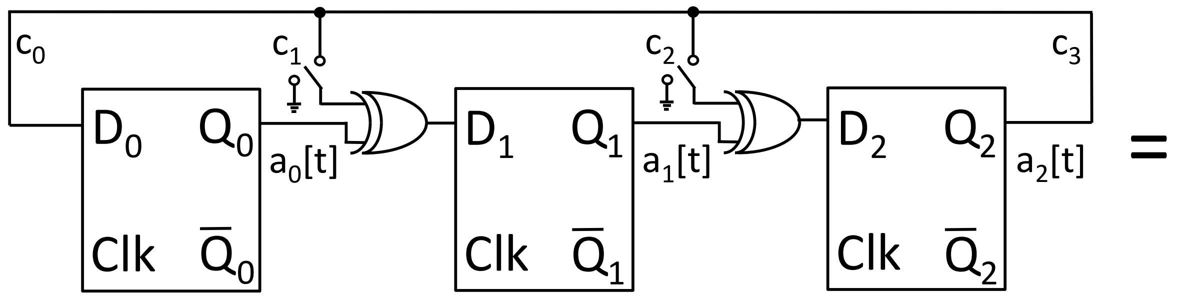
Modulo by the characteristic polynomial

- Start with [001] = 1, multiply by x each iteration, and mod by characterisitic polynomial x^3+x^2+1.

Operation	Unsimplified	Result after % x^3+x^2+1	Binary
$1 * x$	x	x	010
$x * x$	x^2	x^2	100
$x^2 * x$	x^3	$x^2 + 1$	101
$(x^2 + 1) * x$	$x^3 + x$	$x^2 + x + 1$	111
$(x^2 + x + 1) * x$	$x^3 + x^2 + x$	$x + 1$	011
$(x + 1) * x$	$x^2 + x$	$x^2 + x$	110
$(x^2 + x) * x$	$x^3 + x^2$	1	001



Linear Feedback Shift Register (LFSR) Example 5



Example 3: Consider an LFSR (with the notation as defined) with n = 3, c_2=0 and c_1=0.

LFSR:

$P(x) = x^3 + 1$

$U = x$

Input = $\langle \boldsymbol{a_2(t), a_1(t), a_0(t)} \rangle$

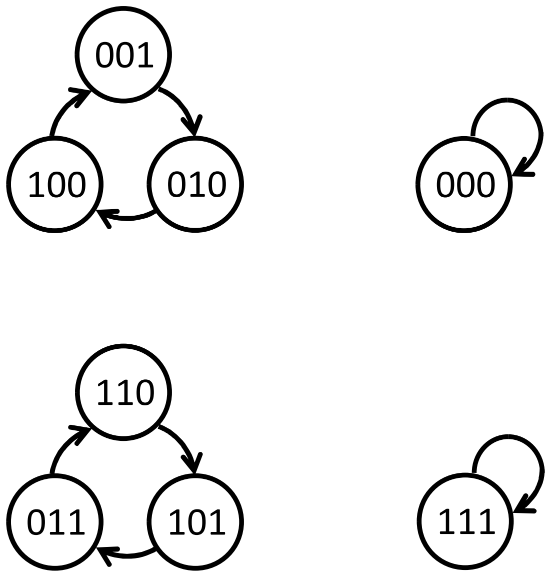
Output = $\langle \boldsymbol{a_2(t+1), a_1(t+1), a_0(t+1)} \rangle$

000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
000	**010**	**100**	**110**	**001**	**011**	**101**	**111**

Modulo by the characteristic polynomial

- Start with [001] = 1, multiply by x each iteration, and mod by characterisitic polynomial x^3+1.

Operation	Unsimplified	Result after % x^3+1	Binary
1 * x	x	x	010
x * x	x^2	x^2	100
x^2 * x	x^3	1	001
(x + 1) * x	$x^2 + x$	$x^2 + x$	110
(x^2 + x) * x	$x^3 + x^2$	$x^2 + 1$	101
(x^2 + 1) * x	$x^3 + x$	$x + 1$	011
(x^2 + x + 1) * x	$x^3 + x^2 + x$	$x^2 + x + 1$	111



LFSR Polynomial Notation

- An LFSR implements multiplication in a polynomial field
- Let *A(t)* be the current state of the LFSR
- *A(t)* = $\langle a_2, a_1, a_0 \rangle$ which can be represented as $a_2x^2 + a_1x + a_0$
- Let the feedback polynomial be *P(x)*
- *A(t+1)* = (*xA(t)*) (mod *P(x)*)

Polynomial Modulus

- Polynomial space
  - E.g., for a 3-bit register, polynomial values range over $a_2x^2 + a_1x + a_0$ where $a_i \in \{0,1\}, 0 \leq i \leq 2$
  - Each $x^i, 0 \leq i \leq 2$ is a position placeholder, i.e., each x^i maintains a position (as opposed to a numerical value)
  - The values $\langle a_2, a_1, a_0 \rangle = \langle 0,0,0 \rangle$ repeat themselves (i.e., for the current state of $\langle 0,0,0 \rangle$ the next state is also $\langle 0,0,0 \rangle$), so this state is referred to as an *absorbing* state and is not considered in the analysis of total number of states reachable by the LFSR
- Two polynomials *a(x)* and *b(x)* are congruent modulo *m(x)* if they leave the same remainder upon division by *m(x)*; this may be expressed as *a(x)* ≡ *b(x)* (mod *m(x)*)

Irreducible and Primitive Polynomials

- A polynomial $x^n + c_{n-2}x^{n-1} + \cdots + c_1x^2 + c_0x + 1$ is said to have *order n*
- A polynomial *a(x)* of order *n* is said to be irreducible if there does not exist any *b(x)* of order i, $1 \leq i \leq n-1$, such that *a(x)* / *b(x)* yields remainder zero
- A major question is when does a characteristic polynomial *P(x)* yield a full period of $2^n - 1$ for an *n*-bit LFSR; a necessary but not sufficient condition is that the characteristic polynomial be irreducible
- Theorem: assuming nonzero initial state, an autonomous LSFR's period is the smallest integer *k* for which *P(x)* evenly divides $x^k + 1$ (i.e., there is no remainder)
- Given that the zero state in an autonomous *n*-bit LFSR repeats infinitely, the maximum size of a periodic state sequence is $2^n - 1$
- Definition: a primitive polynomial yields a full period of $2^n - 1$
- Clearly, then, a primitive polynomial *P(x)* evenly divides $x^{2^n-1} + 1$

Examples

- Consider Example 3: *n* = 3, c_1 = 0 and c_0 = 0 ⇒ *P(x)* = $x^3 + 1$
- If we try *k* = 2, the result is $x^k + 1$ = $x^2 + 1$ and *P(x)* does not evenly divide $x^2 + 1$
- If we try *k* = 3, the result is $x^3 + 1$ and clearly *P(x)* divides itself and so the maximum period is 3! However, since *n* = 3, the maximum period possible is larger (specifically, $2^n - 1$ = 7)
- So consider Example 2: *n* = 3, c_1 = 1 and c_0 = 0 ⇒ *P(x)* = $x^3 + x^2 + 1$
- If we try *k* = {2,3,4,5,6}, they all fail (i.e., the remainder is nonzero)
- If we try *k* = 7, the result is $x^7 + 1$, and $\frac{x^7+1}{P(x)} = x^4 + x^3 + x^2 + 1$ with no remainder
- Clearly, *P(x)* = $x^3 + x^2 + 1$ is primitive!

How Many and How to Generate?

- There is a known mathematical formula for the number of primitive polynomials; the formula uses the Euler totient function $\phi(n)$ [Handbook of Applied Cryptography (HAC) by Menezes, et al.]
- For example, for n = 32, 2^n = $2^{32} = 4{,}294{,}977{,}296$, but there are 67,108,864 primitive polynomials
- To test for a primitive polynomial, one approach is to simulate the full state sequence of $2^n - 1$, but this is not possible for large n
- Another approach to test for a primitive polynomial requires $2^n - 1$ to have a known factorization
- If $2^n - 1$ does not have a known factorization, there is no known efficient (i.e., polynomial time or faster) algorithm to test if the smallest k for which *P(x)* divides $x^k + 1$ with no remainder is when $k = 2^n - 1$
- In other words, given a candidate *P(x)* of order n, if $2^n - 1$ does not have a known factorization, then there is no known efficient algorithm to test if *P(x)* is a primitive polynomial [HAC, pg. 157]

LFSR implements polynomial modulus

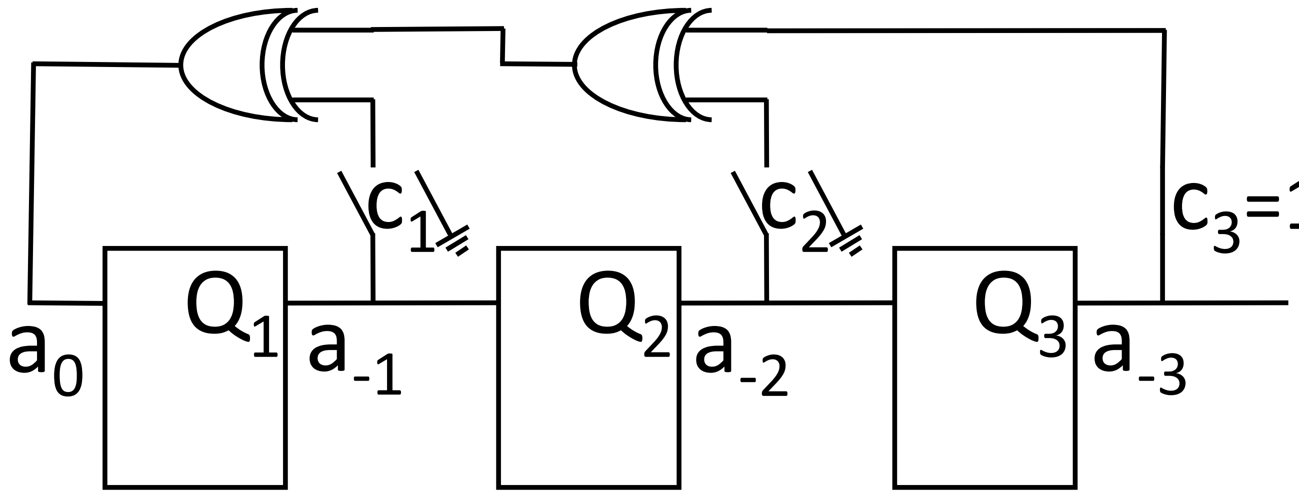
- *A(t+1)* = (*xA(t)*) (mod *P(x)*)
- This is a known result
- If the characteristic polynomial *P(x)* is a primitive polynomial, then the period of the n-bit LFSR is $2^n - 1$, i.e., is full period

Impact Scenarios

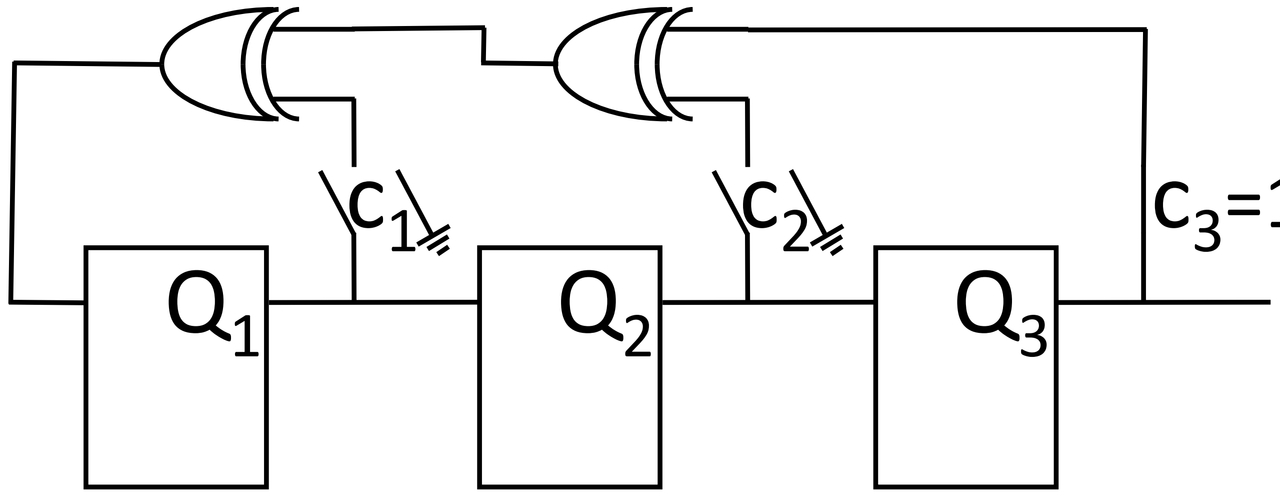
- Smart cards
- Low power IoT devices
- Trusted Platform Module (TPM) design
- Bump-in-the-wire interfacing devices in the power grid

Some Comments on the Notation

- The first bit input to the first FF (with output Q_1) is a_0 at time zero
- The output of the first FF is the input to the first FF at the previous time step; if the current clock step is zero, let's call the previous time step -1; hence the output of the first FF at time zero is a_{-1}

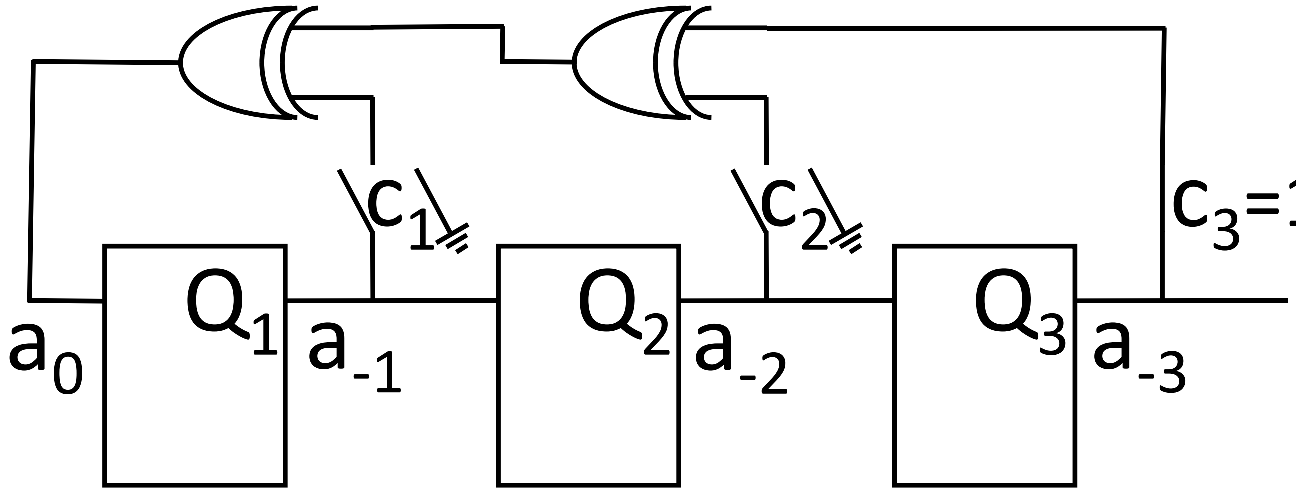


Configurable Linear Feedback Shift Register



Initial State

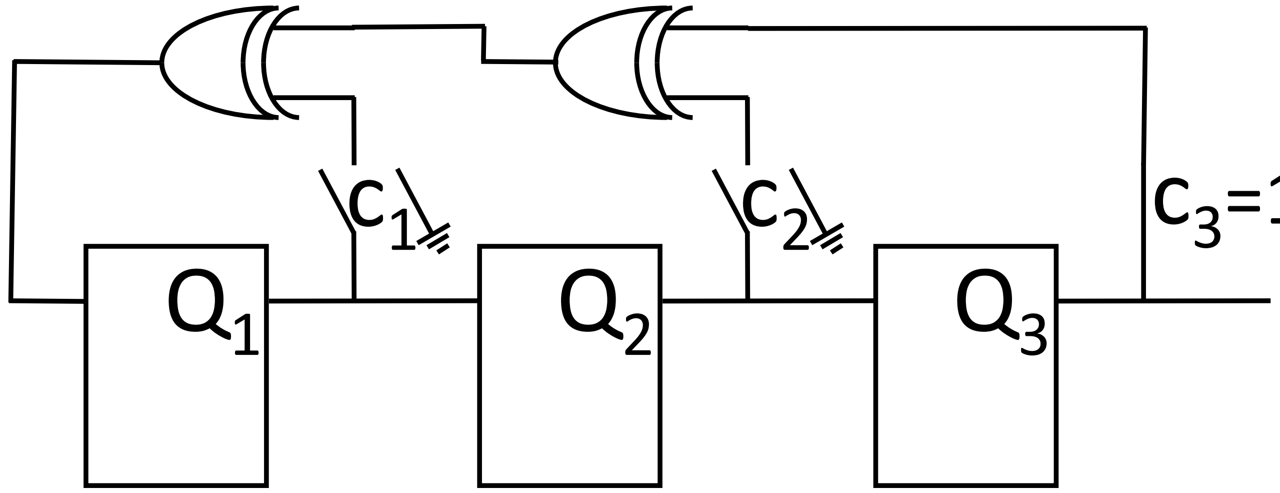
- The input to the first FF is a bit value which is a function
- Each FF output defines a bit



- How many distinct values can the three output bits exhibit?

A Degenerate Case

- Suppose the initial state is all zeros, what happens each clock cycle?



A Question

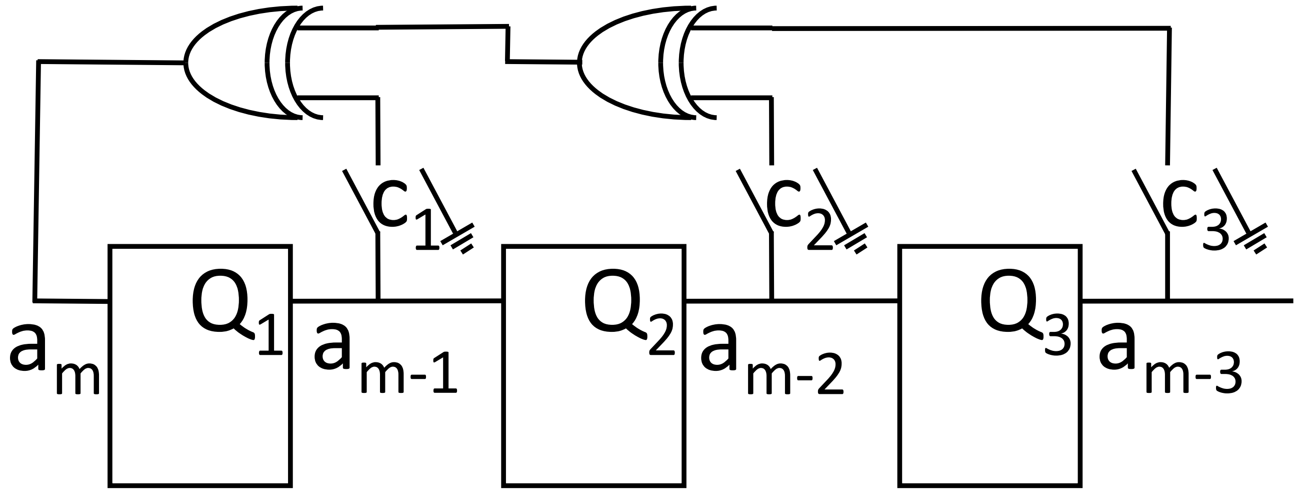
- We must omit the initial state (IS) of all zeros from this design
- We also must omit any state of all zeros – otherwise the state will never change again!
- So for this design we cannot choose c_1, c_2 and c_3 such that all $2^n = 2^3 = 8$ states are used; can we achieve $2^n - 1 = 7$?

An Answer

- Yes!
- The *characteristic polynomial* associated with the LFSR must be a so-called *primitive polynomial*
- We will not cover the theoretical details of these polynomial describing an LFSR
- But you should be able to work out examples of LFSR bit sequences

Current State

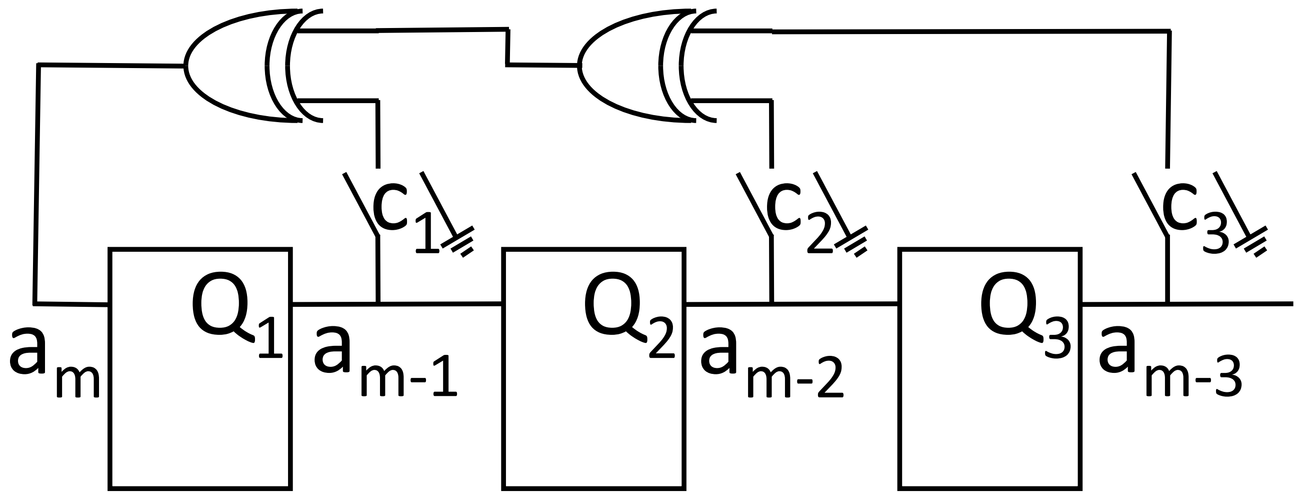
- Clock cycle m



- For n FFs, the sequence is a_m, a_{m-1}, a_{m-2}, …, a_{m-n}

Addition Modulus 2

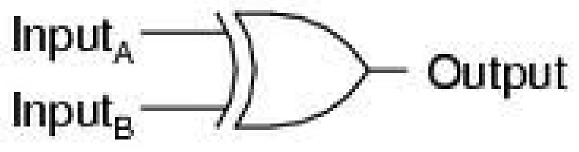
- Note that binary addition modulus 2 becomes XOR

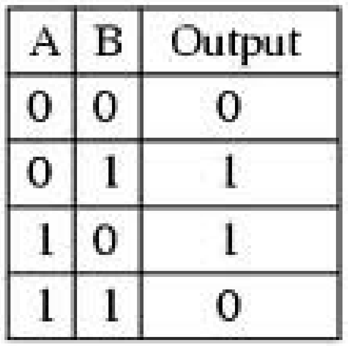


- $a_m = c_1a_{m-1} + c_2a_{m-2} + c_3a_{m-3} \pmod 2 = c_1a_{m-1} \oplus c_2a_{m-2} \oplus c_3a_{m-3}$
- Similarly, binary multiplication becomes AND

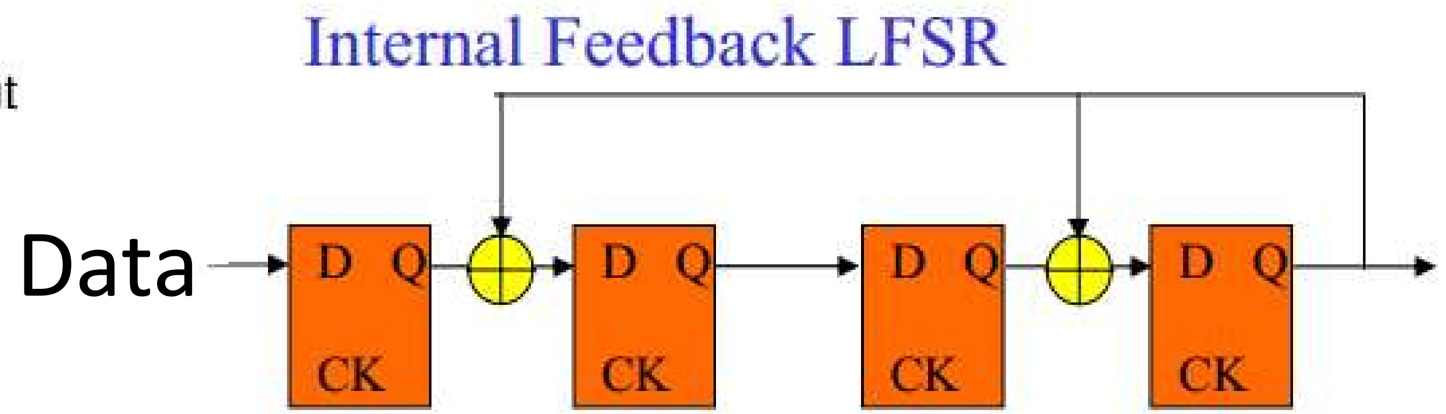
“Internal-XOR” Linear Feedback Shift Register

Exclusive-OR gate

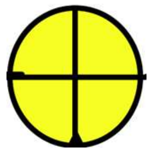




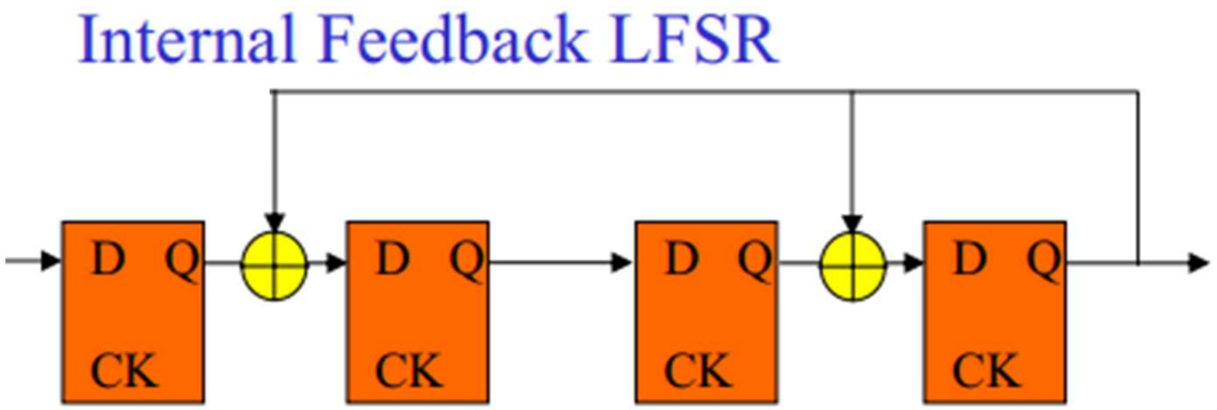
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



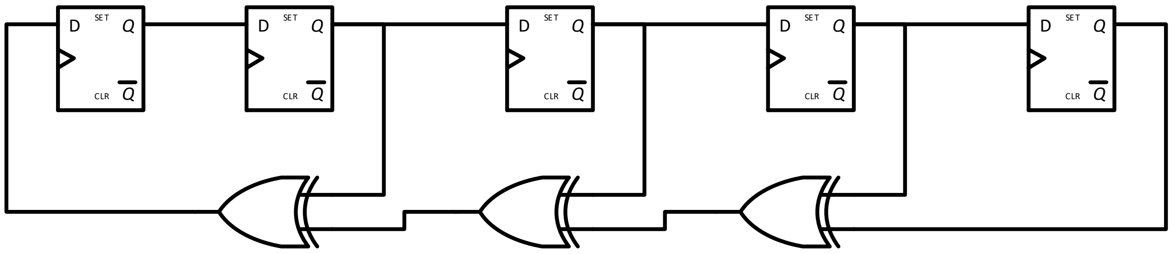
Linear Feedback Shift Register Example Input



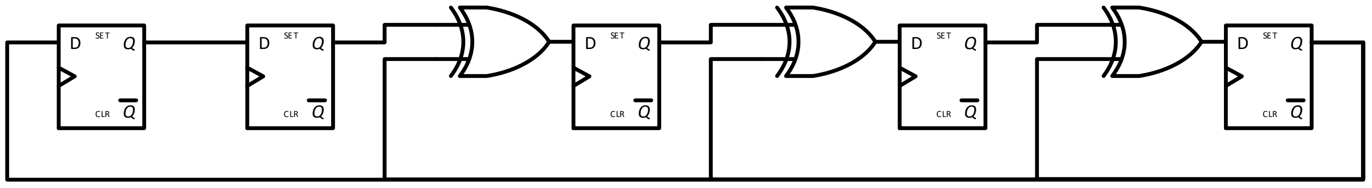
111110101011010



Initial State	**0**	**1**	**1**	**0**
Cycle 1	**0**	**0**	**1**	**1**
Cycle 2	**1**	**1**	**0**	**0**



Fibonacci Type (1) LFSR, with $P(x) = x^5 + x^3 + x^2 + x + 1$



Galois Type (2) LFSR, with same $P(x) = x^5 + x^3 + x^2 + x + 1$

Shift Register Based Hash Function: LFSR

Linear Feedback Shift Registers (LFSRs):

- Are composed of XOR gates and D-flip-flops,
- Can be used as pseudorandom sequence generators,
- Do polynomial division,
  - Where H is a polynomial with coefficients in GF(2) representing the final register state, I is another such polynomial representing the initial state, and P is another such polynomial representing the feedback function:
  - **H** = (**x** · **I**) mod **P**
- Have maximum period when the feedback polynomial is primitive,
  - an irreducible polynomial that that divides $x^{2^\wedge n-1} + 1$, but not $x^d + 1$ for any d that divides $2^n - 1$
- And can be Fibonacci (internal-XOR) or Galois (external-XOR) type
  - Galois is typically faster to implement in software
  - Galois output is the reverse of Fibonacci output

Galois Field

- Finite field with p^n elements where p is a prime number and n is a positive integer
- Usual operations on integers, then mod p
- GF(2)

$$\text{Mod-2 } (x^n + x^n = x^n - x^n = 0)$$

Binary Operations

Mod-2 ($x^n + x^n = x^n - x^n = 0$)

Addition/Subtraction

$$(x^5 + x^2 + 1) + (x^4 + x^2)$$

$$\begin{array}{lllll}
 & x^5 & & x^2 & 1 \\
+ & & x^4 & x^2 & \\
\hline
 & x^5 & x^4 & & 1
\end{array}$$

$$= x^5 + x^4 + 1$$

Multiplication

$$(x^2 + x + 1) \times (x^2 + 1)$$

$$\begin{array}{r}
x^2 + x + 1 \\
\times \quad x^2 + 1 \\
\hline
x^2 + x + 1 \\
x^4 + x^3 + x^2 \qquad\qquad \\
\hline
= x^4 + x^3 + x + 1
\end{array}$$

Division

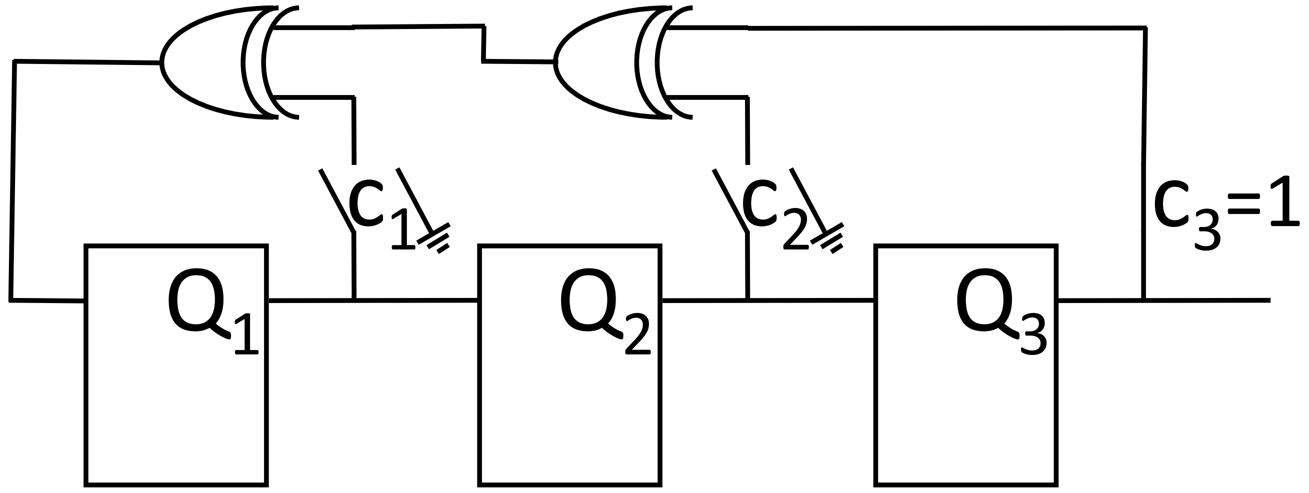
$$\begin{array}{r|l}
 & \quad x^2 + x + 1 \\
\hline
x^2 + 1 & x^4 + x^3 + x + 1 \\
 & x^4 \quad + x^2 \\
\cline{2-2}
 & \quad x^3 + x^2 + x + 1 \\
 & \quad x^3 \qquad + x \\
\cline{2-2}
 & \qquad\qquad x^2 + 1 \\
 & \qquad\qquad x^2 + 1 \\
\cline{2-2}
 & \qquad\qquad\quad 0
\end{array}$$

Example 6:

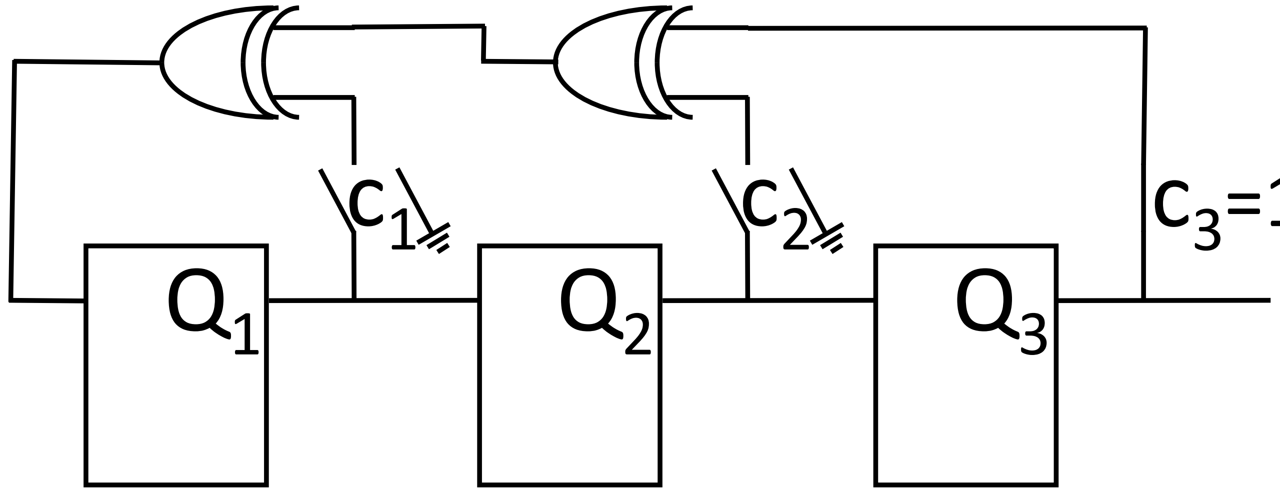
Generating Function

- With no external input, we have an *autonomous LFSR*
- We can associate each a_i with a distinct coefficient in a polynomial
- We use variable *x* in the polynomial, but *x* is never assigned any value
- In a way, *x* keeps track of time where the power indicates the clock cycle (except for zero which indicates the input to the first FF)
- G(x) is an infinite sequence
- $G(x) = a_0 + a_1x^1 + a_2x^2 + \ldots + a_mx^m + \ldots = \sum_{m=0}^{\infty} a_m x^m$
- Note that there is an initial sequence where a_0 works its way from the input to the output; this initial sequence is n clock cycles long

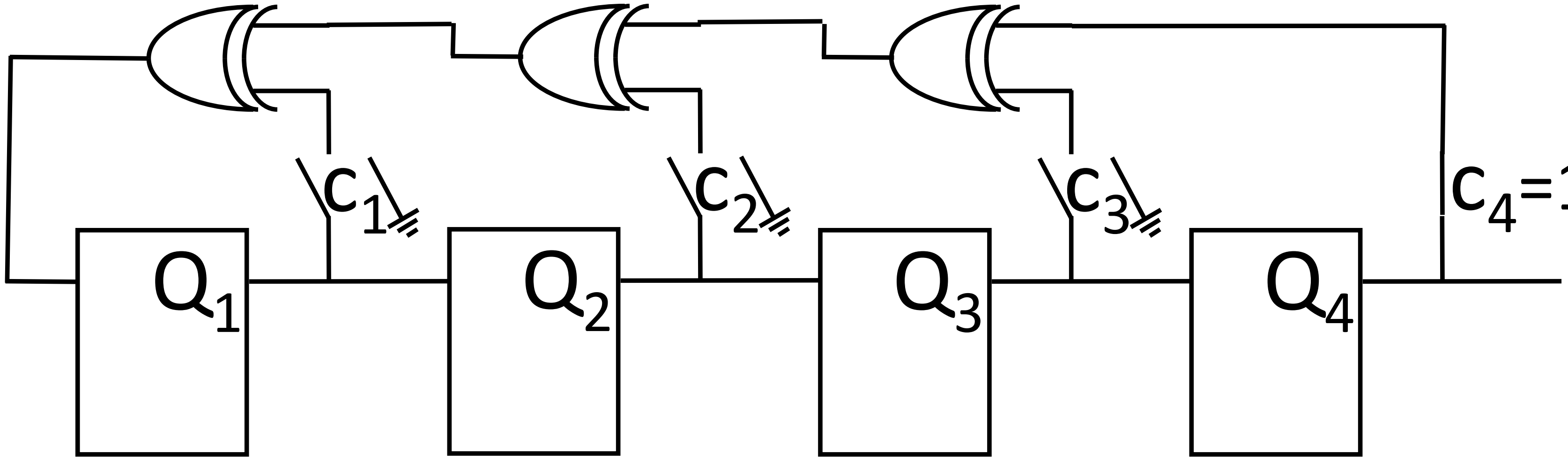
Example 7



Example 8



Example 9



Some Mathematical Results

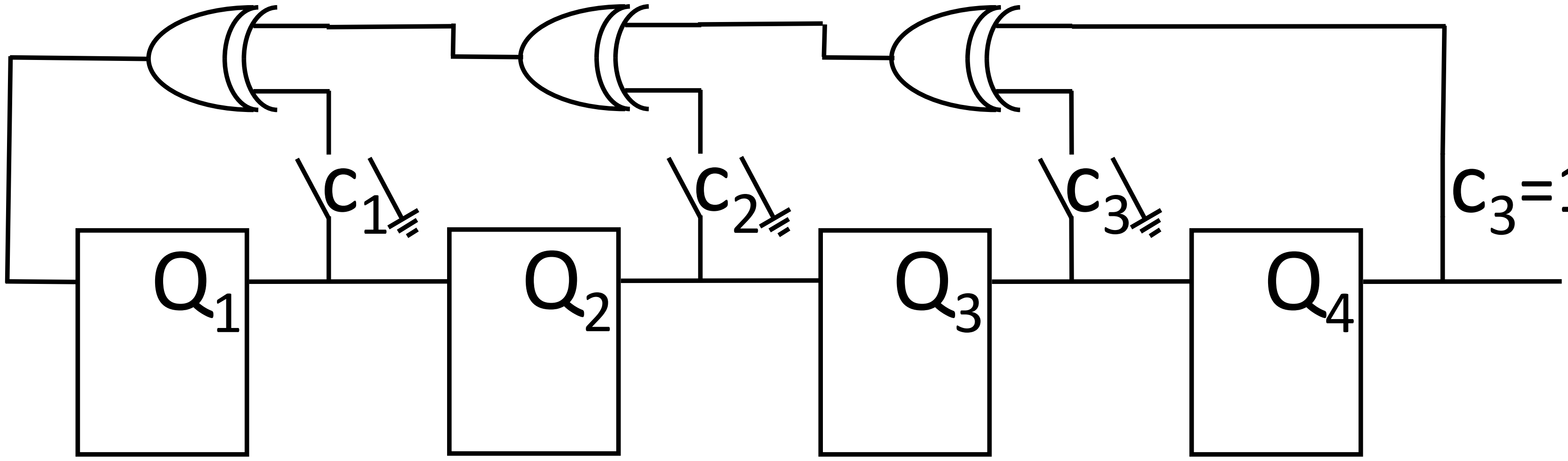
- $a_m = \sum_{i=1}^{n} c_i\, a_{m-i}$
  - E.g., if n = 3 then $a_m = c_1 a_{m-1} + c_2 a_{m-2} + c_3 a_{m-3}$
- $G(x) = \sum_{m=0}^{\infty} a_m x^m = \sum_{m=0}^{\infty} \left[\sum_{i=1}^{n} c_i\, a_{m-i}\right] x^m = \sum_{i=1}^{n} c_i x^i \sum_{m=0}^{\infty} a_{m-i} x^{m-i}$
- $= \sum_{i=1}^{n} c_i x^i \left[a_{-i} x^{-i} + \cdots + a_{-1} x^{-1} + \sum_{m=0}^{\infty} a_m x^m\right]$
- $= \sum_{i=1}^{n} c_i x^i \left[a_{-i} x^{-i} + \cdots + a_{-1} x^{-1} + G(x)\right]$
- $= \sum_{i=1}^{n} c_i x^i G(x) + \sum_{i=1}^{n} c_i x^i \left(a_{-i} x^{-i} + \cdots + a_{-1} x^{-1}\right) = G(x)$
- $\Rightarrow G(x) = \frac{\sum_{i=1}^{n} c_i x^i \left(a_{-i} x^{-i} + \cdots + a_{-1} x^{-1}\right)}{1 + \sum_{i=1}^{n} c_i x^i}$, which for $a_{-i} = 0$ except $a_{-n} = 1$
- $\Rightarrow G(x) = \frac{c_n x^n \left(a_{-n} x^{-n}\right)}{1 + \sum_{i=1}^{n} c_i x^i} = \frac{1}{1 + \sum_{i=1}^{n} c_i x^i}$ since $c_n = 1$ always for an n-bit LFSR
- The denominator is referred to as the *characteristic polynomial* of the sequence:
- $P(x) = 1 + c_1 x + c_2 x^2 + \ldots + c_n x^n$

Some Mathematical Results (Continued)

- recall $G(x) = \sum_{m=0}^{\infty} a_m x^m$ = $a_0 + a_1x + a_2x^2 + \ldots + a_mx^m + \ldots$
- and if we choose initial state (IS) $a_{-1} = 0$, $a_{-2} = 0$, …, $a_{-(n-1)} = 0$, $a_{-n} = 1$
- $\Rightarrow G(x) = \frac{1}{1+\sum_{i=1}^{n} c_i x^i} = \frac{1}{1+c_1x+c_2x^2+\cdots+c_nx^n} = \frac{1}{P(x)}$ and note G(x) periodic (say, p)
- $\Rightarrow \frac{1}{P(x)}$ = $(a_0 + a_1x + a_2x^2 + \ldots + a_{p-1}x^{p-1})$ +

  $x^p(a_0 + a_1x + a_2x^2 + \ldots + a_{p-1}x^{p-1}) + x^{2p}(a_0 + a_1x + a_2x^2 + \ldots + a_{p-1}x^{p-1}) + \ldots$
- $\Rightarrow \frac{1}{P(x)}$ = $(a_0 + a_1x + a_2x^2 + \ldots + a_{p-1}x^{p-1})(1 + x^p + x^{2p} + \ldots) = \frac{a_0+a_1x+a_2x^2+\cdots+a_{p-1}x^{p-1}}{1-x^p}$

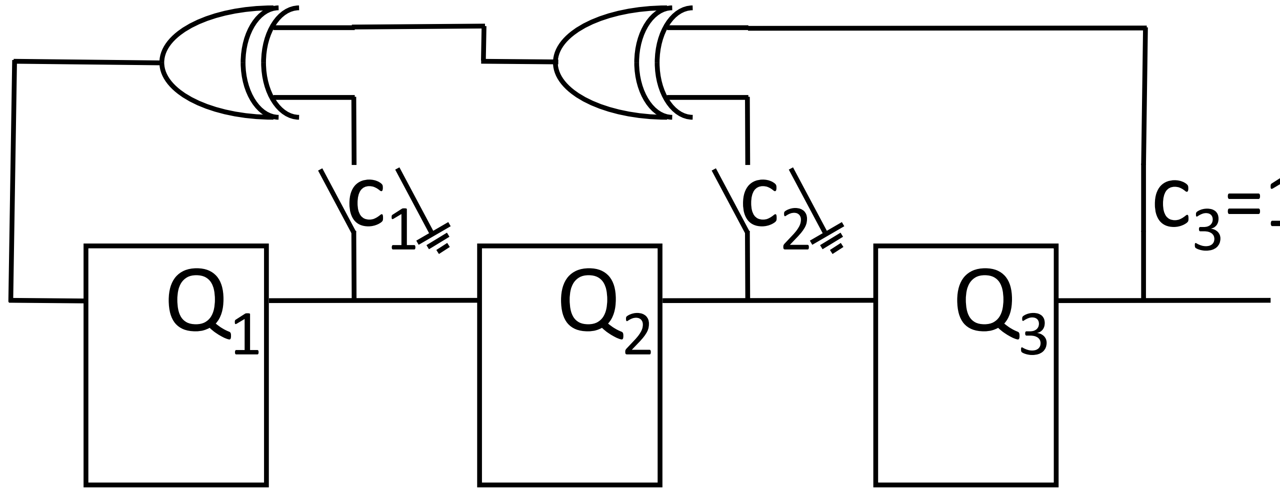
Example 10



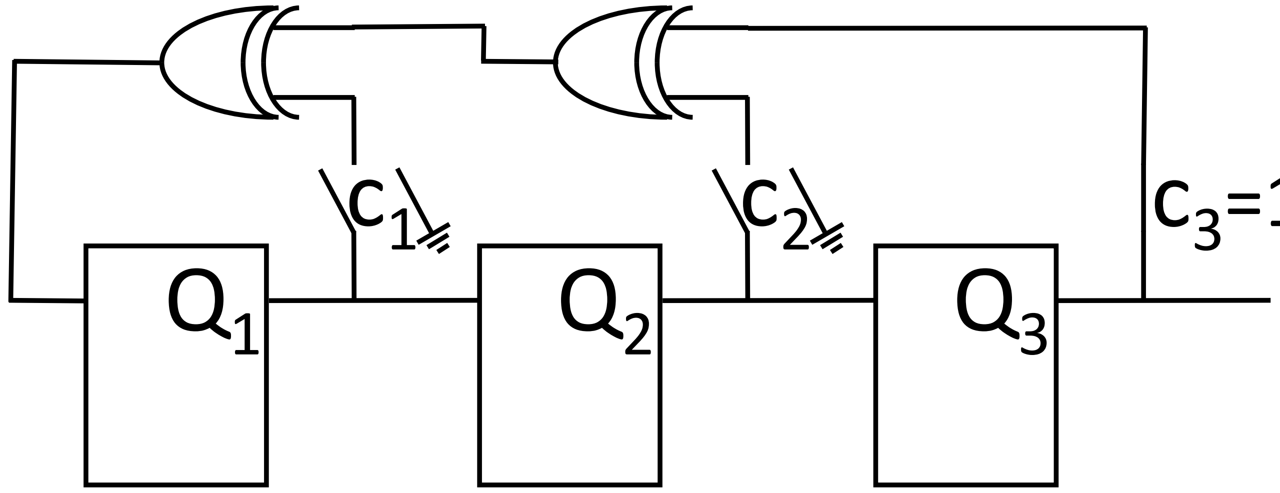
Comments

- There can also be derived what is called the *reciprocal polynomial* $P^*(x) = x^n P(1/x) = c_n + c_{n-1}x + c_{n-2}x^2 + \ldots + c_1 x^{n-1} + x^n$
- As a result, this LFSR can be described by two types of polynomials

Example 11



Example 12

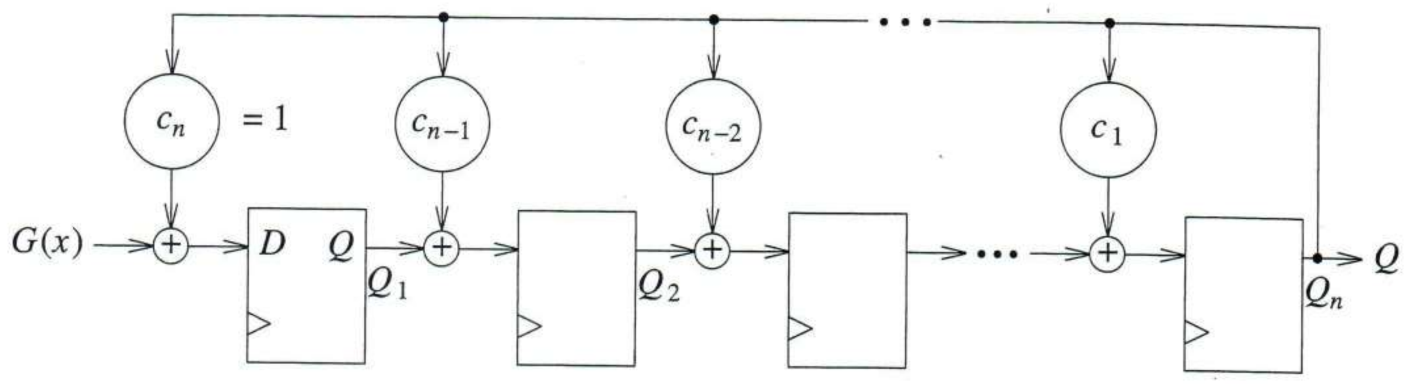


Periodicity

- Maximum length of period p for an LFSR with n FFs is $2^n - 1$
- Theorem: Given an LFSR with initial state $a_{-1} = 0$, $a_{-2} = 0$, …, $a_{-(n-1)} = 0$, $a_{-n} = 1$, the LFSR sequence $\{a_m\}$ is periodic with the smallest integer k for which P(x) divides $(1-x^k)$
- Note that “divides” in the theorem means there is no remainder
- recall $\frac{1}{P(x)} = \frac{a_0+a_1x+\cdots+a_{p-1}x^{p-1}}{1-x^p}$, this implies $\frac{1-x^p}{P(x)} = a_0 + a_1x + \cdots + a_{p-1}x^{p-1}$
- Defn.: If the sequence generated by an LFSR with n FFs has period $2^n - 1$, then it is called a *maximum-length sequence*
- Defn.: the characteristic polynomial associated with a maximum-length sequence is called a *primitive polynomial*

More Comments

- We will not cover how to generate primitive polynomials
- However, the sequences they generate have the following properties:
- Property 1: given a sequence of m bits, the number of ones differs from the number of zeros by at most one
- Property 2: given a sequence of m bits, the number of runs of ones equals the number of runs of zeros
- Property 3: given a sequence of m bits, one half of the runs have length one, one fourth have length two, one eighth have length three, and so forth, as long as the fractions result in integral numbers of runs



$$P(x) = 1 + \sum_{i=1}^{n} c_i x^i$$

Initial state: $I(x) = 0$; Final state: $R(x)$

$$P^*(x) = 1 + \sum_{i=1}^{n} c_i x^{n-i}$$

$$\frac{G(x)}{P^*(x)} = Q(x) + \frac{R(x)}{P^*(x)}$$

or

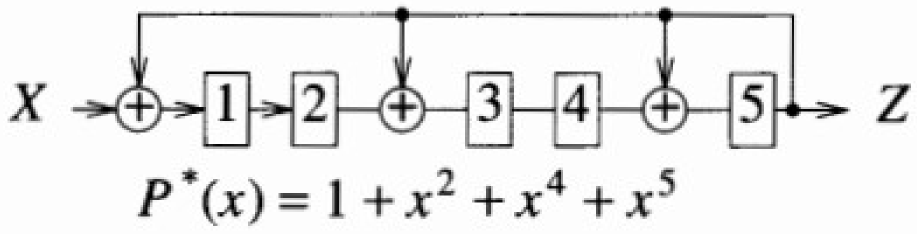
$$G(x) = Q(x)\,P^*(x) + R(x)$$

Figure 10.15 A type 2 LFSR used as a signature analyzer

Signature Analyzers

- LFSRs can be used to compress test bit data (stuck-at fault testing)
- Consider an input of m bits: there are 2^m possible inputs from the testing
- However, the n-bit LFSR can produce a periodic output with 2^n possible out-puts (note the all-zero state is now allowed due to the presence of input)
- It has been shown that the number of bitstreams of length m that produce the same output of length n is $2^m/2^n$
- Thus $2^m/2^n – 1 = 2^{m-n} – 1$ erroneous bitstreams exist that produce the same signature (i.e., there is a test failure but the n-bit output is identical to a passing value)
- Since there are $2^m – 1$ erroneous bitstreams possible, the proportion of erroneous bitstreams exist that produce the same signature is $(2^{m-n} – 1)/(2^m – 1\ ) \cong 2^{-n}$
- For n = 16, $100(1 – 2^{-n}) = 99.9984$ percent

Characteristic Polynomial



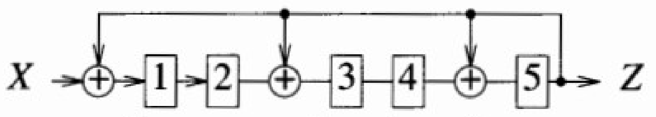
$P^*(x) = 1 + x^2 + x^4 + x^5$

$\longrightarrow$

Input sequence: 1 1 1 1 0 1 0 1 (8 bits)

$G(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1$

$$\sum_{i=1}^{n} c_i x^i a_m(t)$$



$P^*(x) = 1 + x^2 + x^4 + x^5$

Input sequence: $\overrightarrow{1\ 1\ 1\ 1\ 0\ 1\ 0\ 1}$ (8 bits)

$G(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1$

Time	Input stream	Register contents	Output stream
		1 2 3 4 5	
0	1 0 1 0 1 1 1 1	0 0 0 0 0 ←	Initial state
1	1 0 1 0 1 1 1	1 0 0 0 0	
⋮	⋮	⋮	
5	1 0 1	0 1 1 1 1	
6	1 0	0 0 0 1 0	1
7	1	0 0 0 0 1	0 1
8	Remainder →	0 0 1 0 1	1 0 1

$$\underbrace{0\ 0\ 1\ 0\ 1}_{\text{Remainder}} \qquad \underbrace{1\ 0\ 1}_{\text{Quotient}}$$

Remainder $R(x) = x^2 + x^4$ Quotient $1 + x^2$

$$\begin{array}{rl} P^{*}(x): & x^5 + x^4 + x^2 + 1 \\ \times\, Q(x): & x^2 + 1 \\ \hline & x^7 + x^6 + x^4 + x^2 + x^5 + x^4 + x^2 + 1 \\ = & x^7 + x^6 + x^5 + 1 \end{array}$$

Input sequence: 1 1 1 1 0 1 0 1 (8 bits)

$$G(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1$$

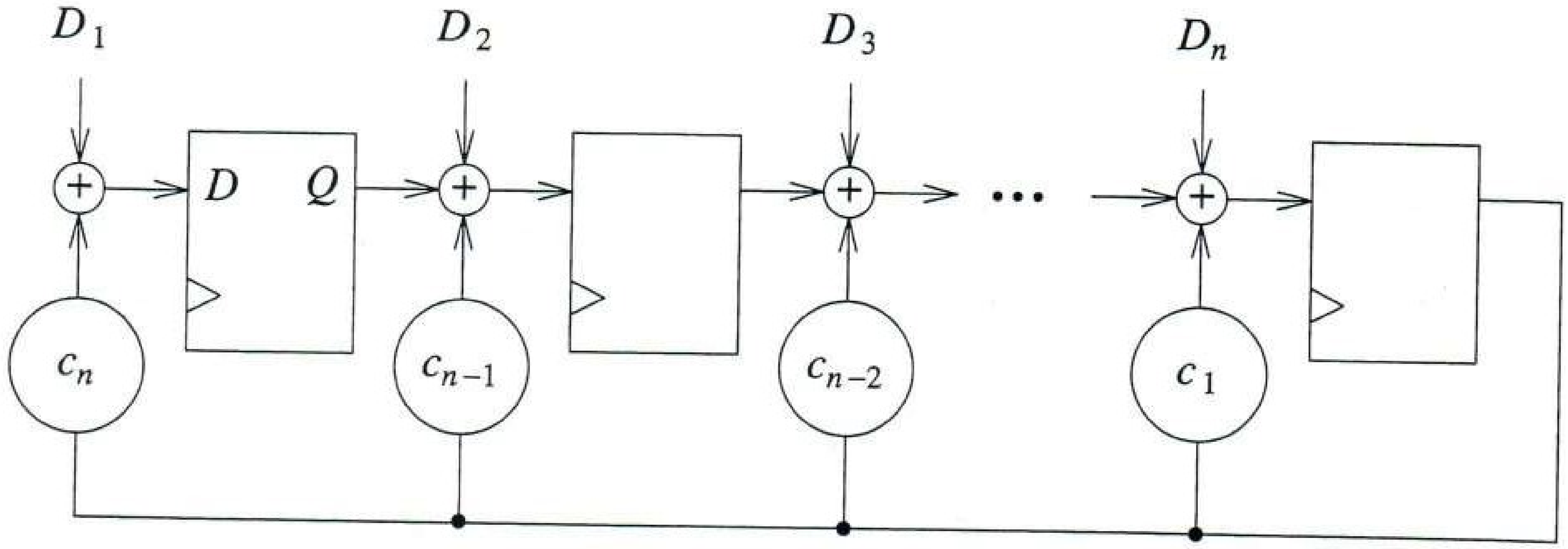
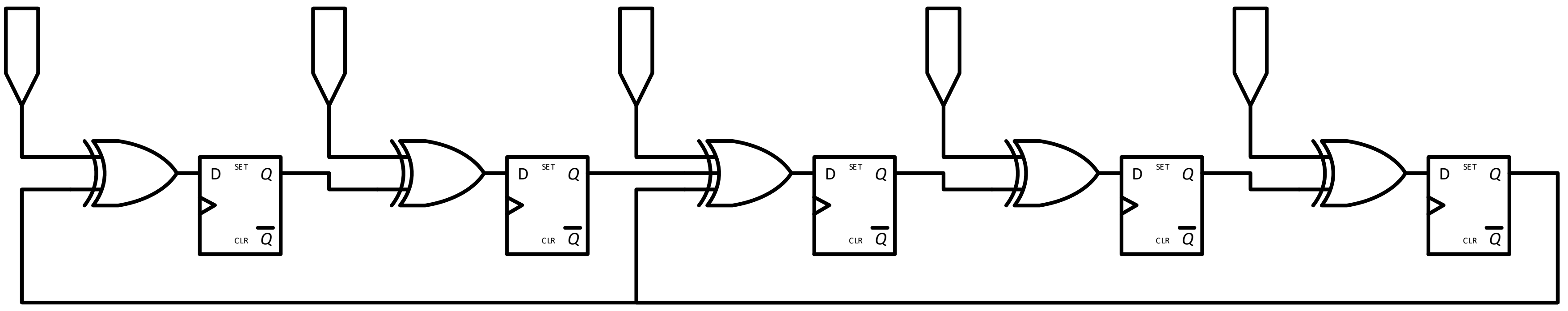


Figure 10.17 Multiple-input signature register

Multiple Input Signature Register (MISR)

- A MISR is an LFSR extended by including another input into each XOR gate
- A MISR can be used as a signature generator by having a message input block by block into the MISR and reading the final register state
- Key:
  - The initial state of the register, and
  - The fact that the MISR performs polynomial division.
    - Where H is a polynomial with coefficients in GF(2) representing the final register state, I is another such polynomial representing the initial state, M is another such polynomial representing the input message, and P is another such polynomial representing the feedback function $H = (x \cdot I + M) \bmod P$

MISR Example



$P(x) = x^5 + x^3 + 1$

Linear Feedback Shift Register Example