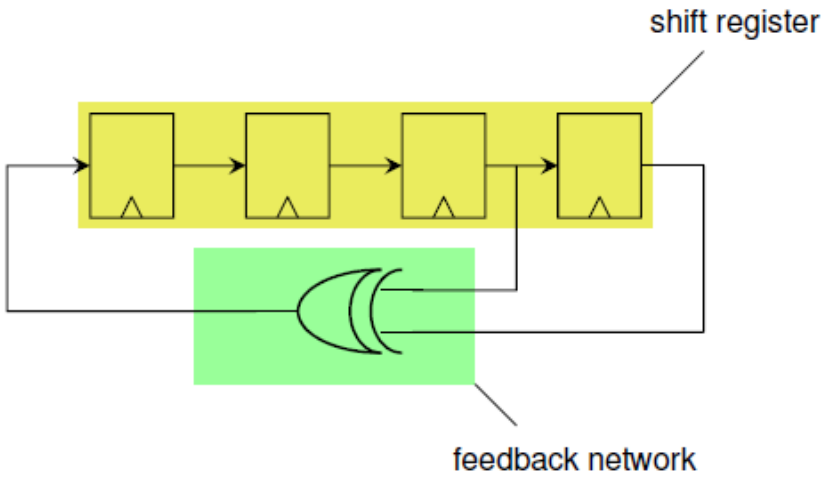
Linear Feedback Shift Registers

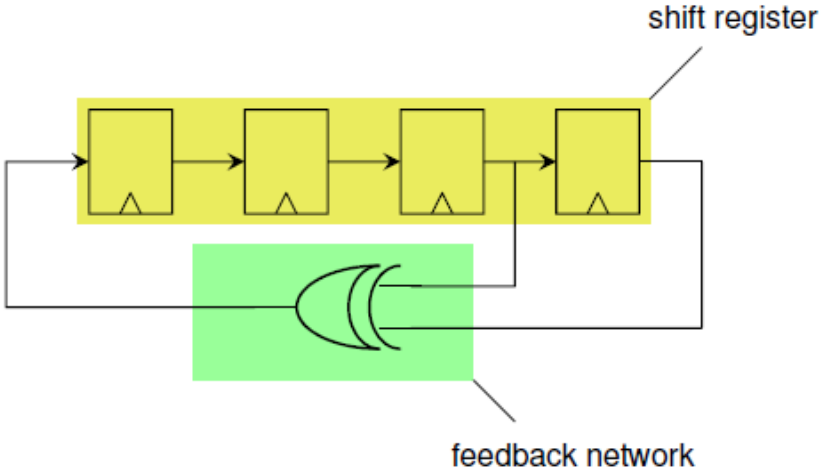
Linear Feedback Shift Registers (LFSR)



- A type of circuit made from XOR and D flip-flop.

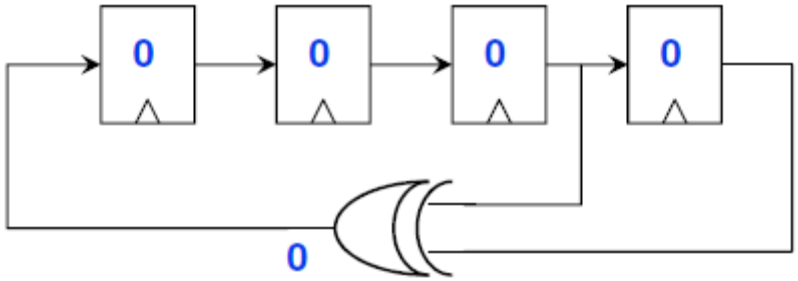
Linear Feedback Shift Register (LFSR)

- A form of the shift register.
- Typical Applications:
  - Fast Counters
  - Built-in Self Test (BIST)
  - Pseudo-random Number Generation
  - Data Encryption and Decryption
  - Data Integrity Checksums
  - Data Compression Techniques
  - Cryptography



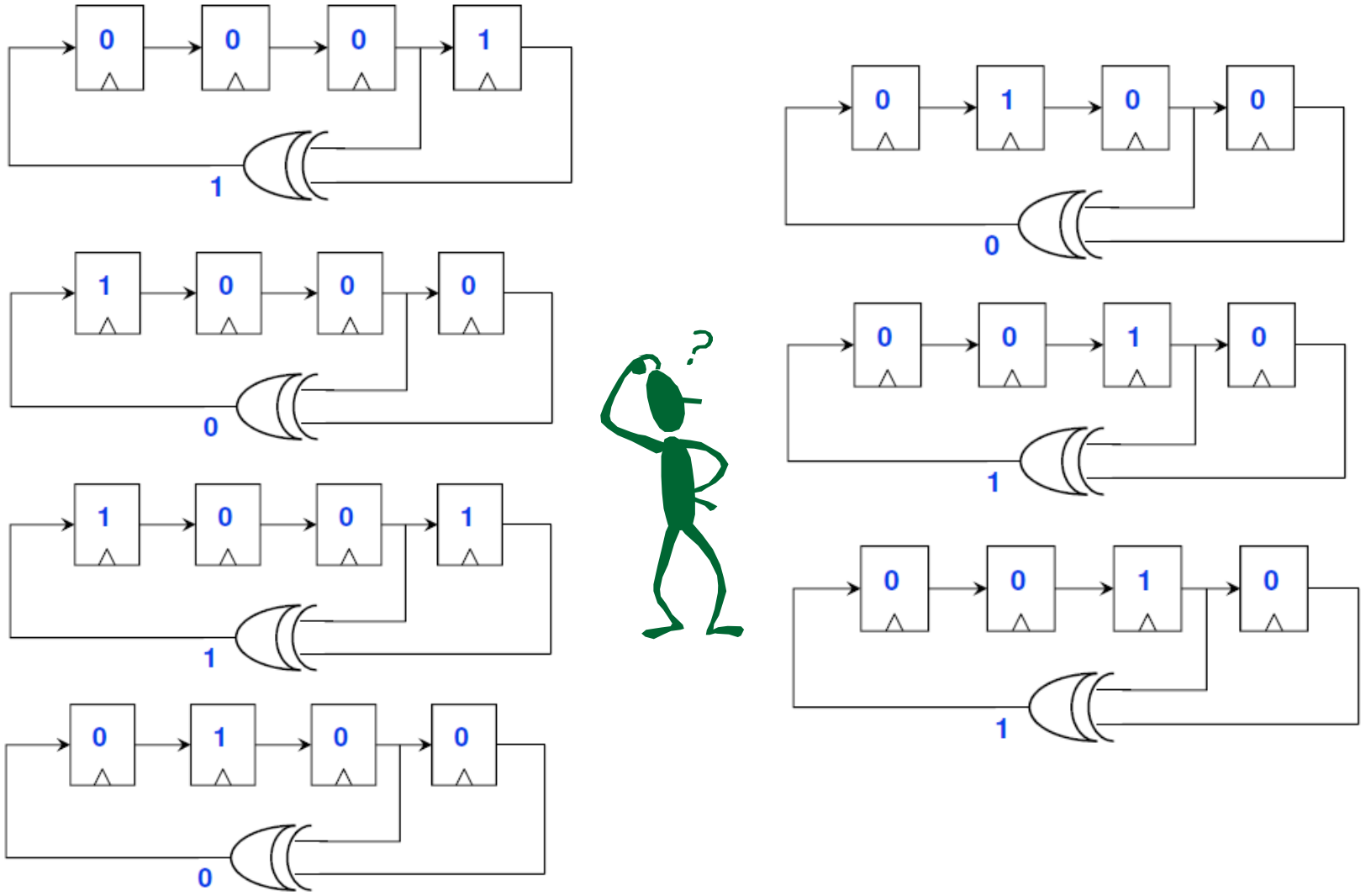
- Both hardware and software implementations of LFSRs are common.

Example 1



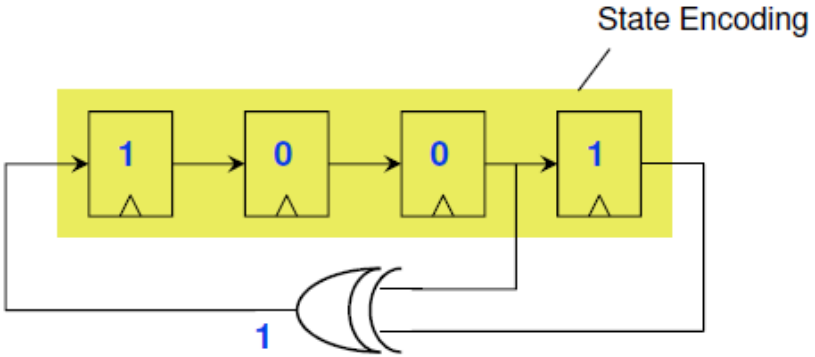
- All zeroes
  - not very useful …

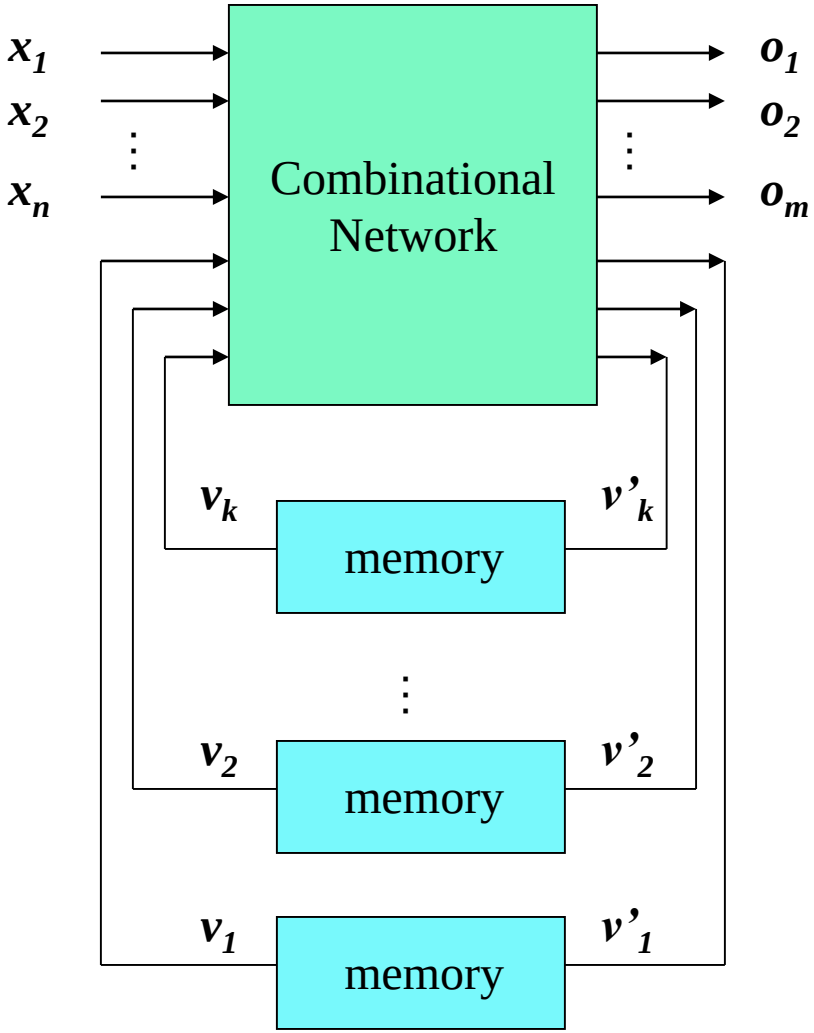
Non-zero Interesting Configurations



Linear Feedback Shift Register

- It is a finite state machine.

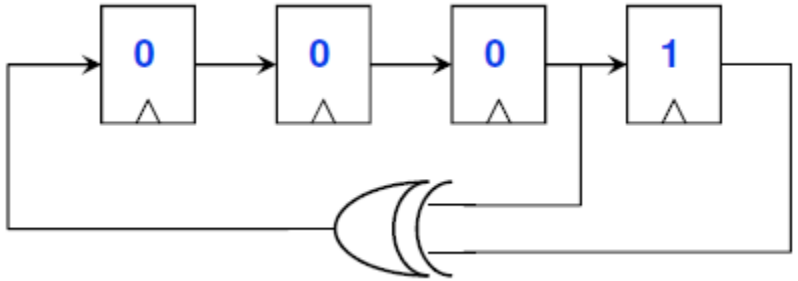


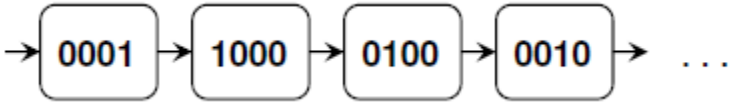


- Current state variables?
- Next state variables?

State Enumeration

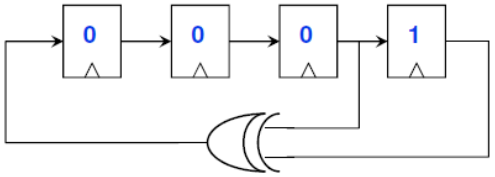
- How many states will you see?

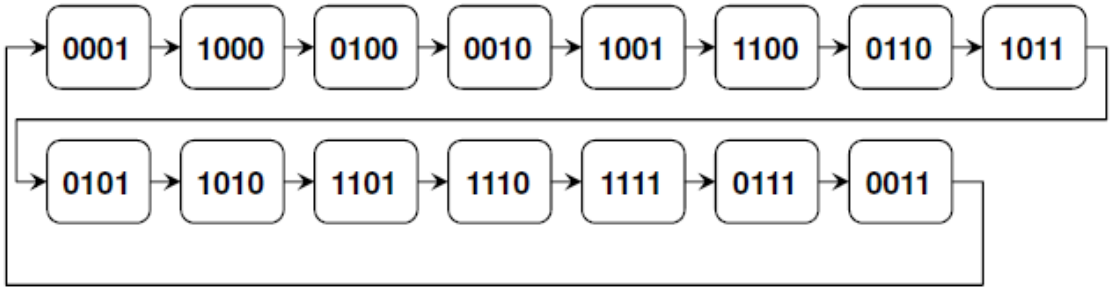




State Space: A Cyclic Structure

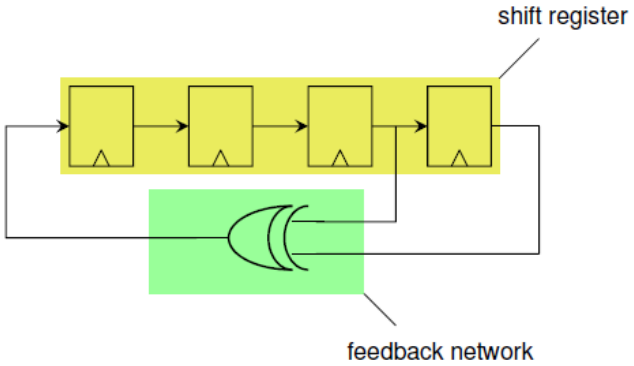
- 15 states.





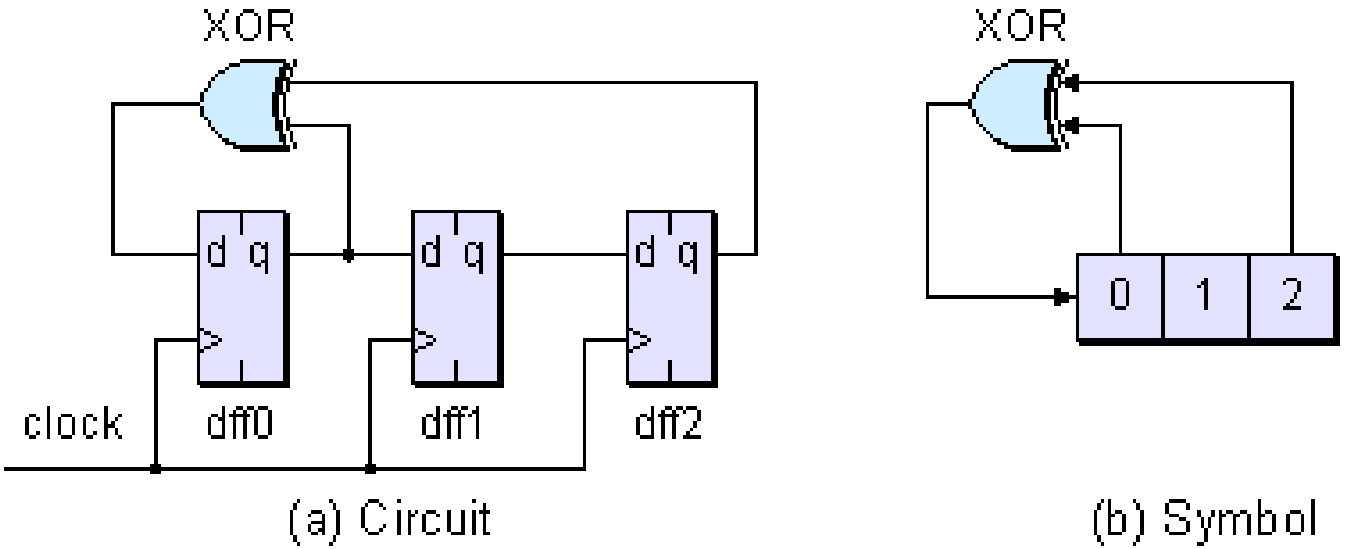
- Because the register has a finite number of possible states,
  - it must eventually enter a repeating cycle.
- However, a LFSR with a well-chosen feedback function can produce a sequence of bits
  - which appears "random" and
  - which has a very long cycle.

Linear Feedback Shift Register



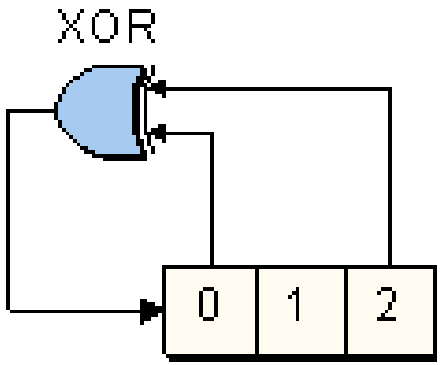
- A linear feedback shift register (LFSR) is a shift register
  - whose input bit is a linear function of its previous state.
- The only linear functions of single bits are xor and inverse-xor;
- It is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the overall shift register value.
- The sequence of values produced by the register is completely determined by its current (or previous) state.

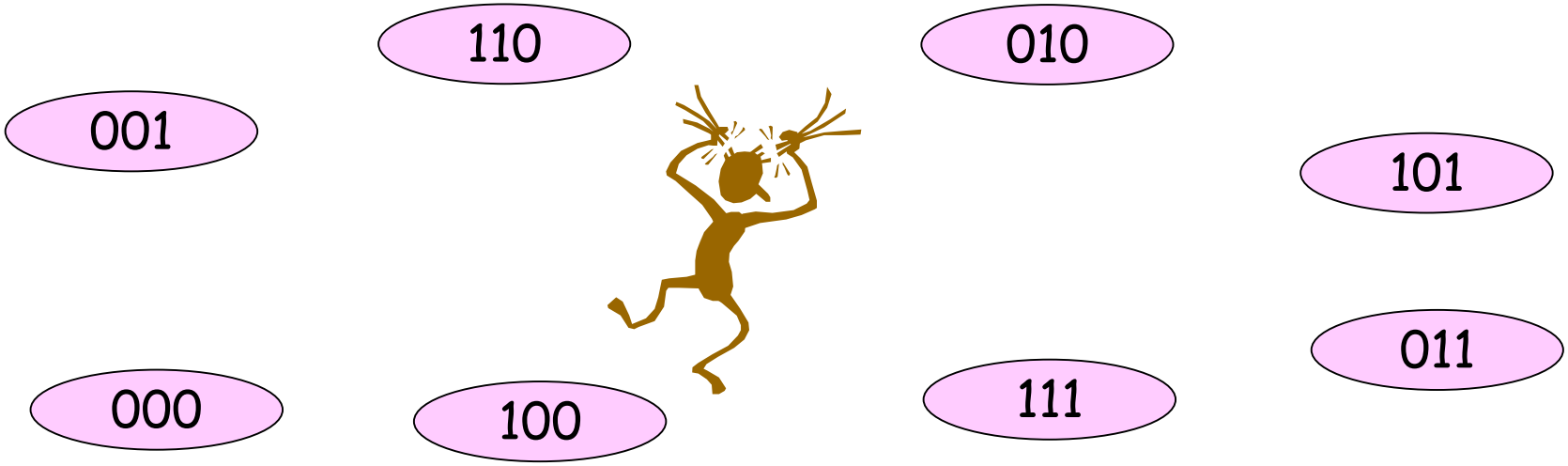
A Simplified Model with Taps



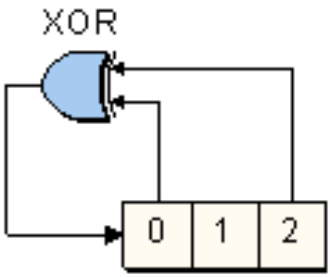
- An LFSR is formed from a simple shift register with feedback from two or more points, or taps, in the register chain.
- The list of the bits positions that affect the next state is called the tap sequence.
  - Example: [bit0, bit2], or [0, 2]

Example 3: Taps [0, 2]

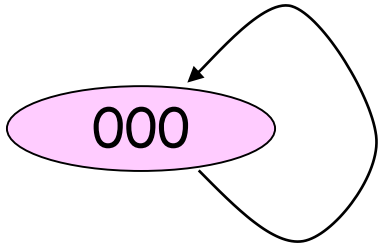


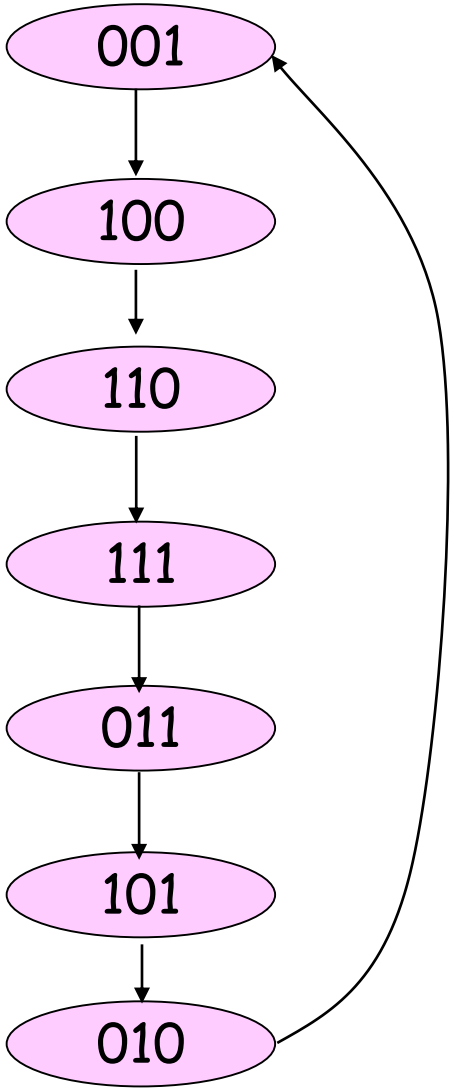


Example 3: Taps [0, 2]

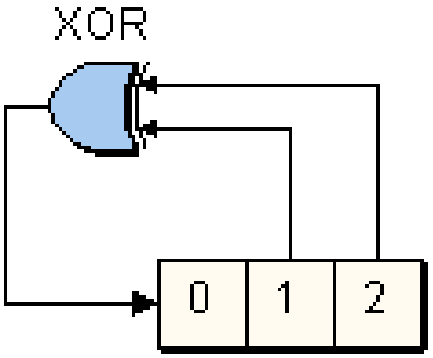


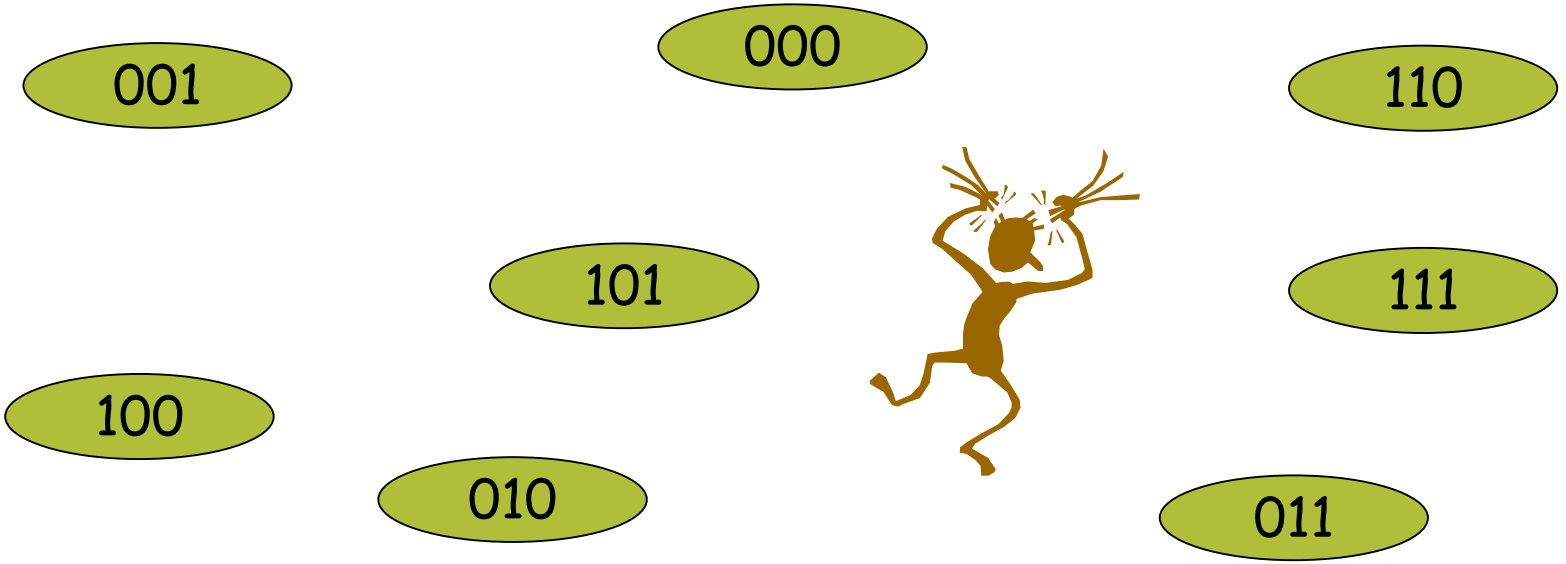
Clock	q0	q1	q2	
--	0	0	1	←
1	1	0	0	
2	1	1	0	
3	1	1	1	
4	0	1	1	
5	1	0	1	
6	0	1	0	
7	0	0	1	
8	1	0	0	
9	1	1	0	
:	:	:	:	



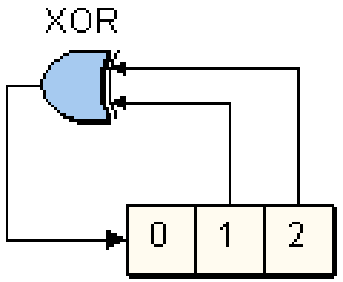


Example 3: Taps [1, 2]

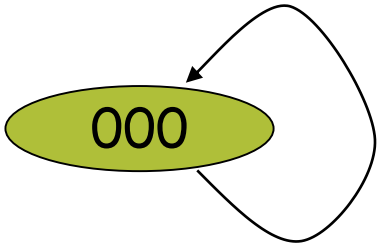


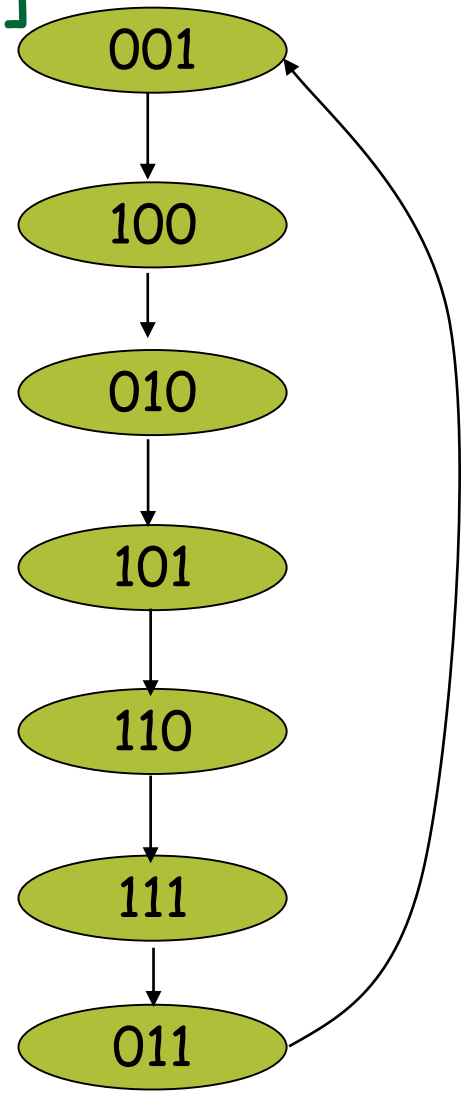


Example 3: Taps [1, 2]

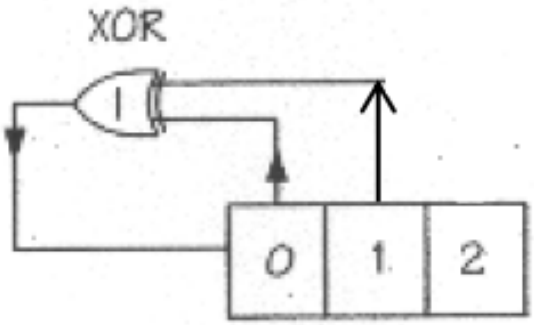


Clock	q0	q1	q2
--	0	0	1
1	1	0	0
2	0	1	0
3	1	0	1
4	1	1	0
5	1	1	1
6	0	1	1
7	0	0	1
8	1	0	0
9	0	1	0
:	:	:	:





Example 3: Taps [0, 1]



001

010

100

000

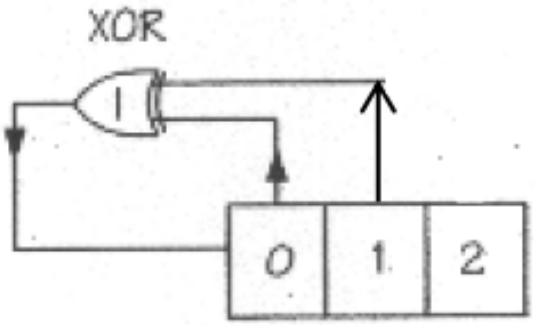
110

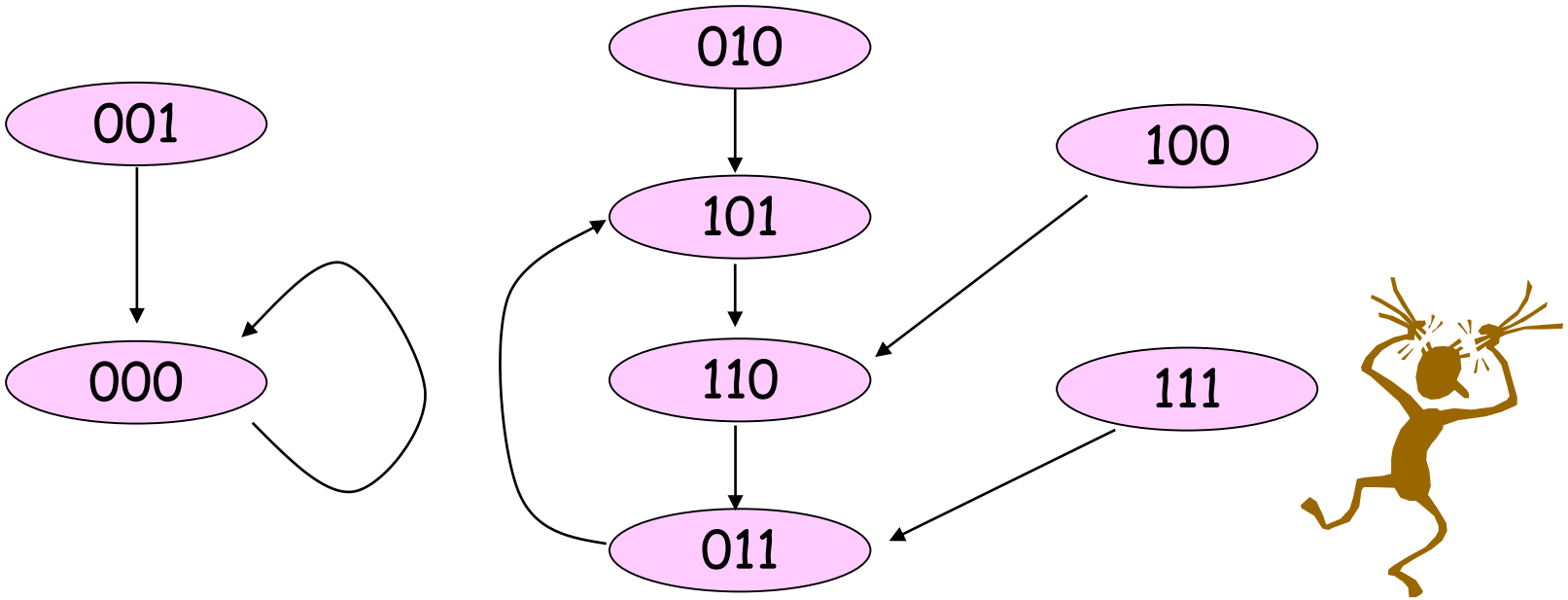
111

101

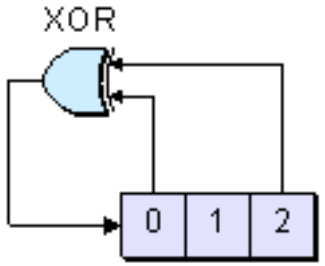
011

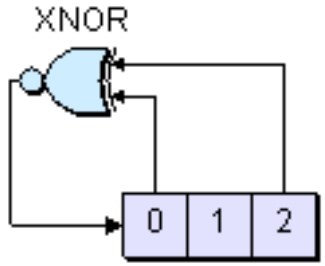
Example 3: Taps [0, 1]





Example 4: Different Feedbacks with Different Sequences

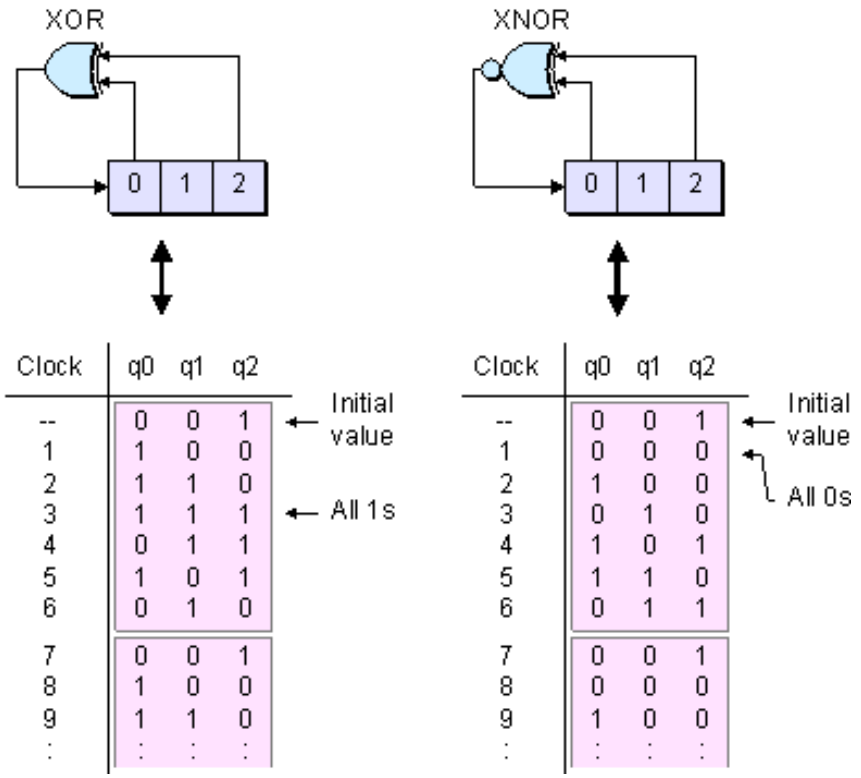




Clock	q0	q1	q2	
--	0	0	1	← Initial value
1	1	0	0	
2	1	1	0	
3	1	1	1	← All 1s
4	0	1	1	
5	1	0	1	
6	0	1	0	
7	0	0	1	
8	1	0	0	
9	1	1	0	
:	:	:	:	

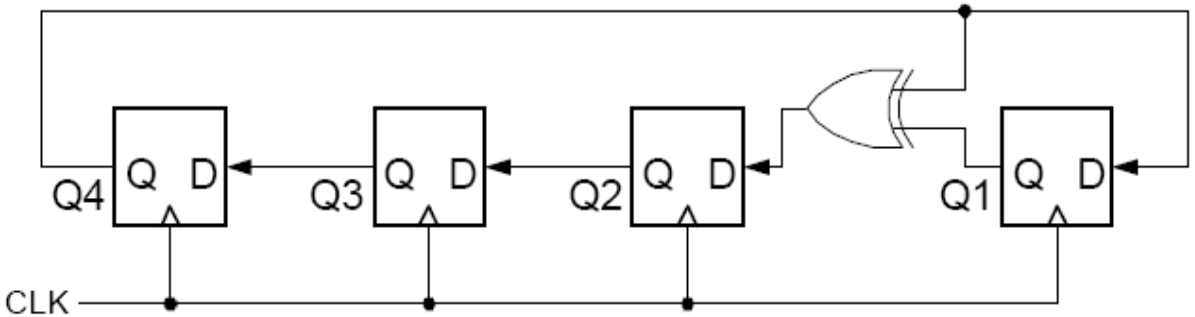
Clock	q0	q1	q2	
--	0	0	1	← Initial value
1	0	0	0	← All 0s
2	1	0	0	
3	0	1	0	
4	1	0	1	
5	1	1	0	
6	0	1	1	
7	0	0	1	
8	0	0	0	
9	1	0	0	
:	:	:	:	

The State Sequence



- The sequence of states generated by an LFSR is determined by
  - its feedback function and tap selection.

Example 3: Internal LFSR



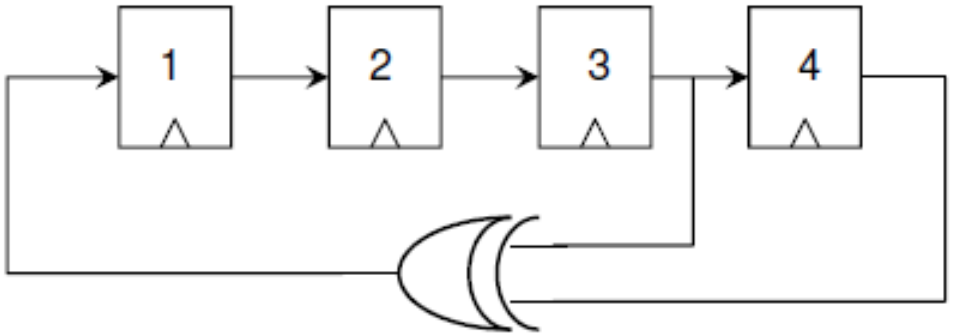

```
    0 0 0 1 0
xor 0 0 0 0 0
    0 0 0 1 0 0
xor 0 0 0 0 0
    0 0 1 0 0 0
xor 0 0 0 0 0
    0 1 0 0 0 0
xor 1 0 0 1 1
    0 0 0 1 1 0
xor 0 0 0 0 0
    0 0 1 1 0 0
xor 0 0 0 0 0
    0 1 1 0 0 0
xor 1 0 0 1 1
    0 1 0 1 1
```

Q4 Q3 Q2 Q1

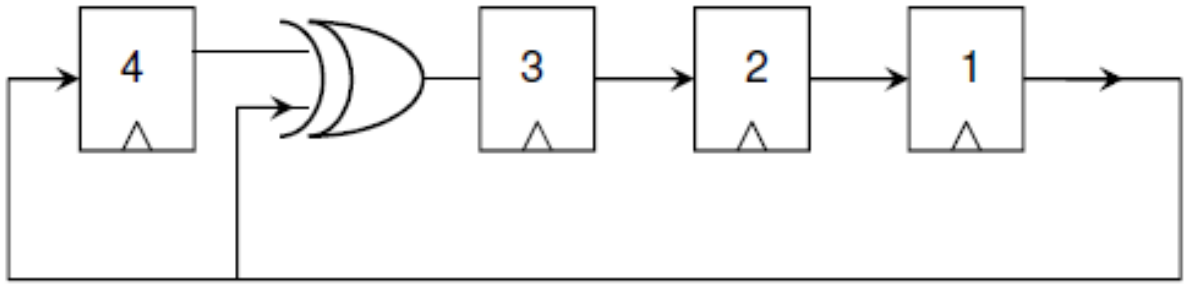
0001
0010
0100
1000
0011
0110
1100
1011
0101
1010
0111
1110
1111
1101
1001
0001

Fibonacci and Galois LFSR

- This format is called a Fibonacci LFSR.



- Can be converted to an equivalent Galois LFSR.

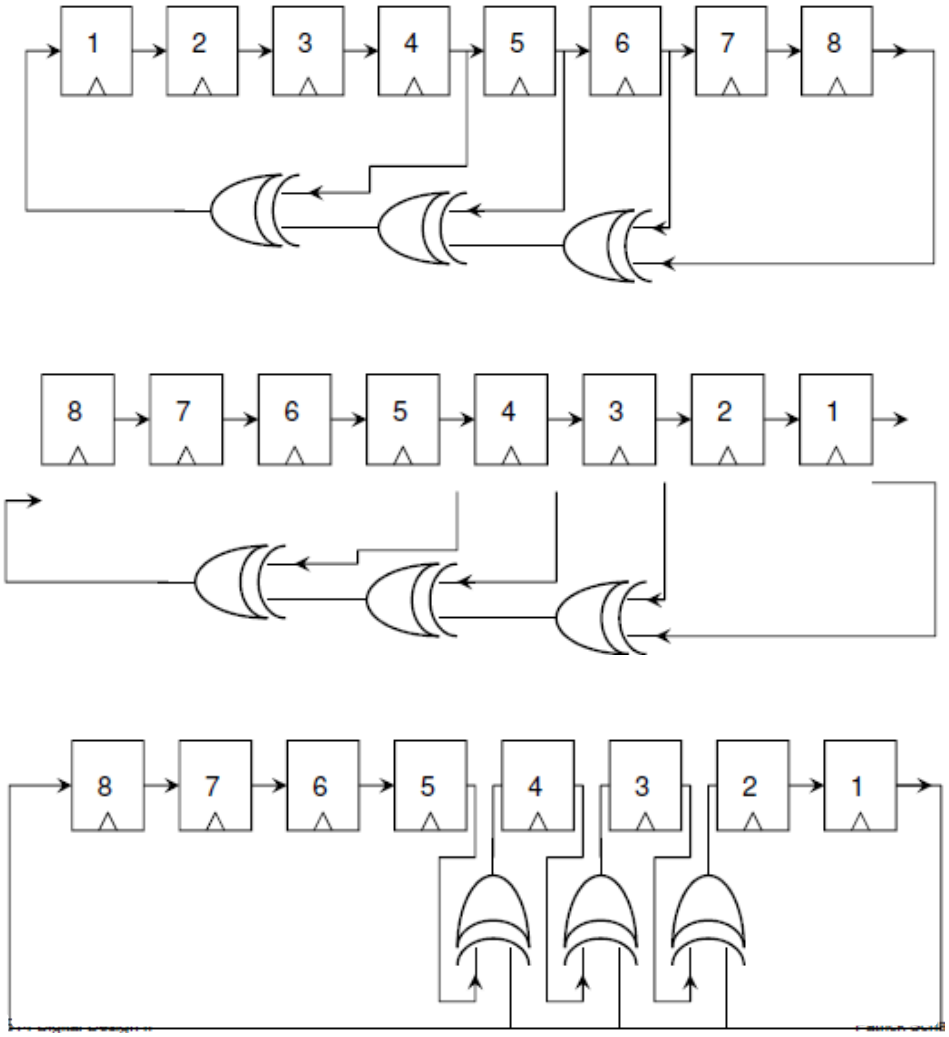


Fibonacci
~1175-1250

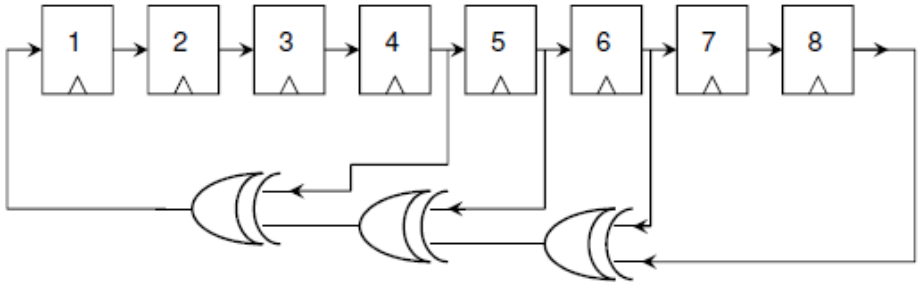
Evariste
Galois
1811-1832

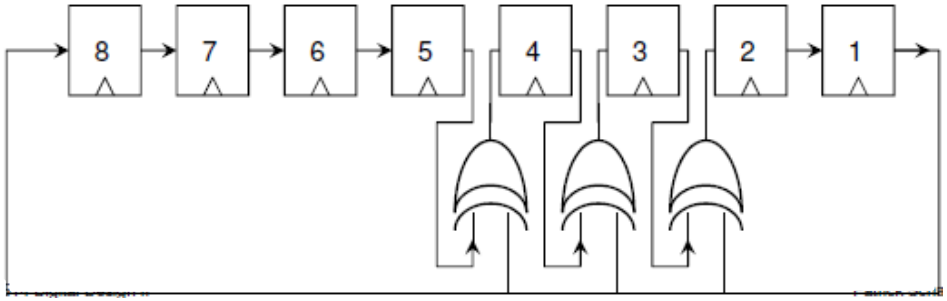
Fibonacci and Galois LFSR

- Each Fibonacci LFSR can transform into Galois LFSR:
  - Reverse numbering of taps
  - Make XOR inputs XOR outputs and vice versa
- Example:
  - Disconnect XOR inputs
  - Reverse tap numbering (not the direction of shifting!)
  - Turn XOR inputs into XOR outputs and vice versa



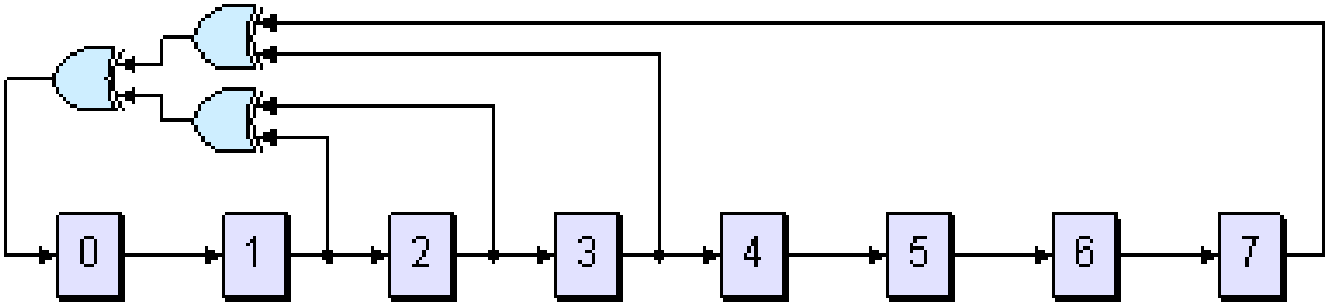
Fibonacci and Galois LFSR



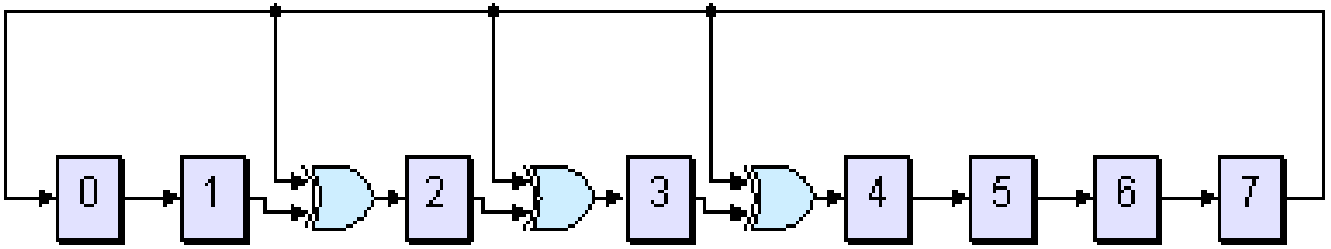


- Which one is better for digital hardware?
- Fibonacci:
  - a chain of XOR
- Galois:
  - computes all taps in parallel

One-to-Many & Many-to-One



(a) Many-to-one implementation

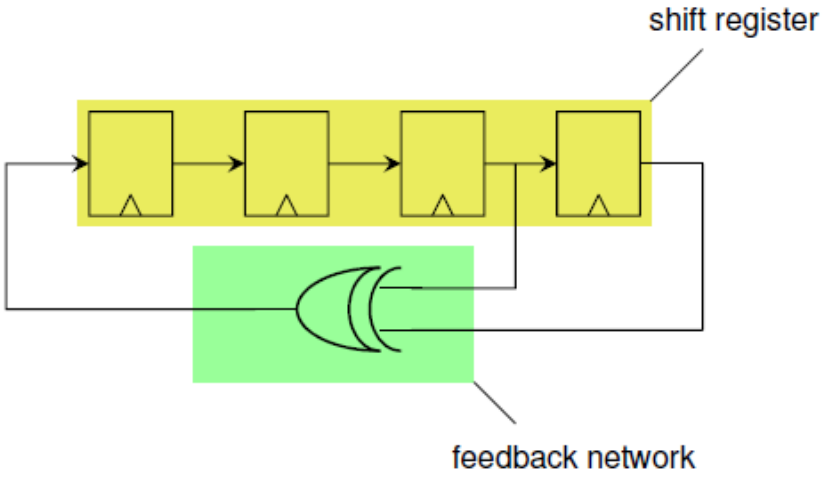


(b) One-to-many implementation

LFSR

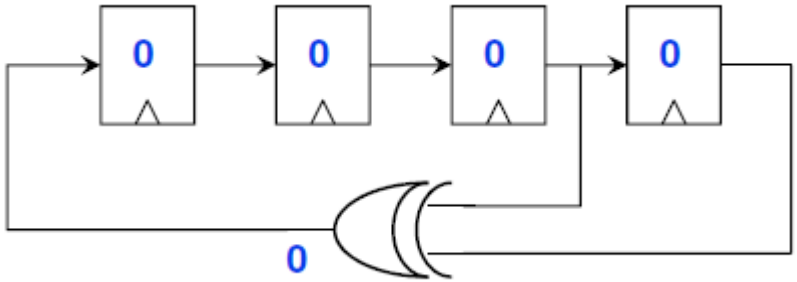
- In general, with n flip-flops, we may have: 2^n-1 different non-zero bit patterns.
- A maximal LFSR produces a sequence
  - cycles through all possible states within the shift register except all zero bits,
  - unless it contains all zeros, in which case it will never change.
- Maximal length:
  - it sequences through every possible value (excluding all the bits being 0) before returning to its initial value.

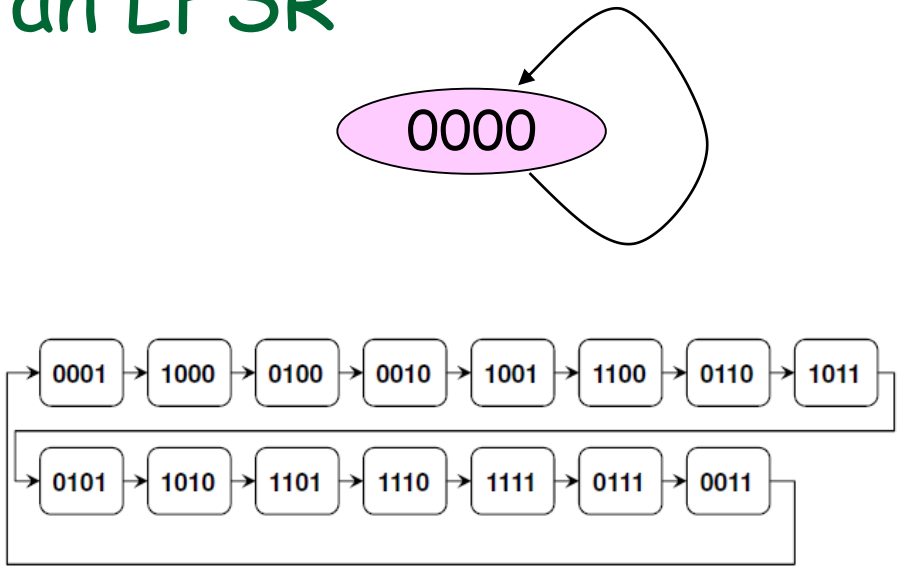
LFSR



- The choice of taps determines
  - how many values there are in a given sequence before the sequence is repeated.
- Depending how the values feedback,
  - the value sequence will not be the same.

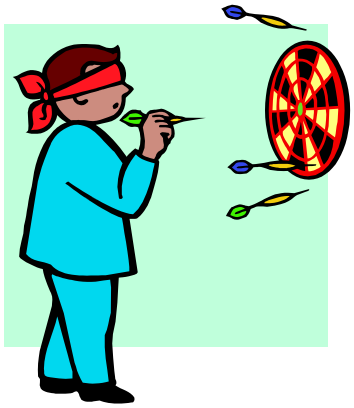
Seeding an LFSR

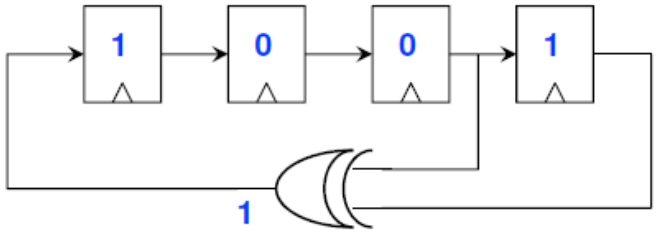


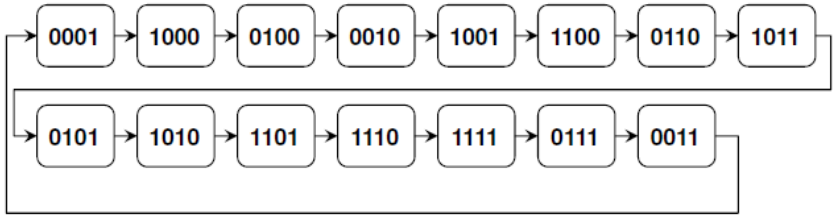


- The initial value of the LFSR is called the seed.
- With an XOR-based LFSR, if it happens to find itself in the all 0s, it will happily continue to shift all 0s indefinitely.
- Similarly, for an XNOR-based LFSR and the all 1s.
- This is of particular concern when power is first applied to the circuit.

Seeding an LFSR

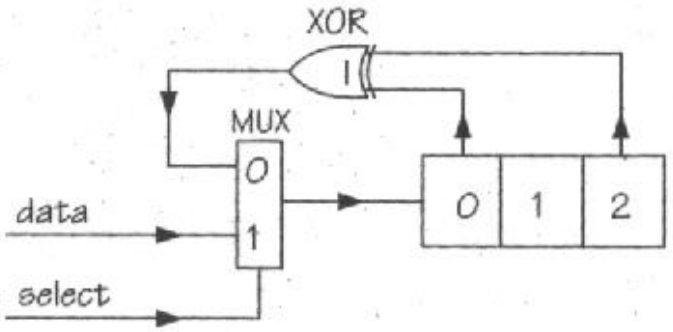


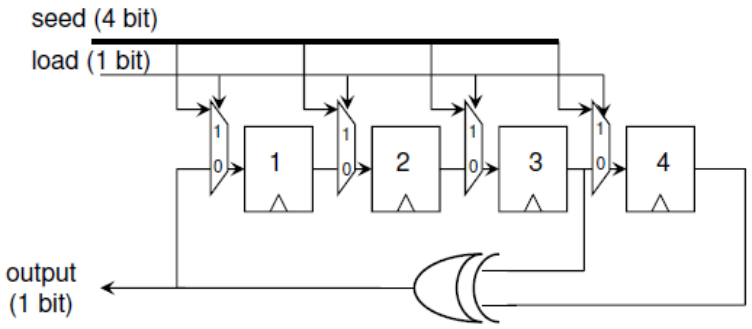




- Each register bit can randomly initialize containing either a logic O or a logic 1,
  - the LFSR can wake up containing its "forbidden" value.
- It is necessary to initialize an LFSR with a seed value.

Seeding an LFSR

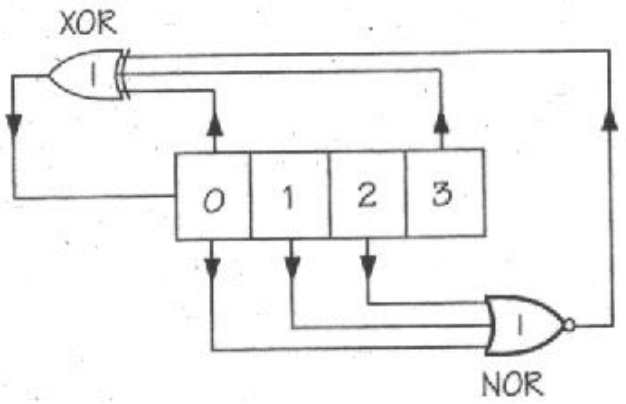




- In certain applications, it is desirable to be able to vary the seed value.
- To include a multiplexer at the input, to the LFSR.
- When the data input is selected, the device functions as a standard shift register and any desired seed value may be loaded.
- After loading the seed value,
  - the feedback path is selected and the device returns to its LFSR mode of operation.

Modifying LFSRs to Sequence 2^n Values

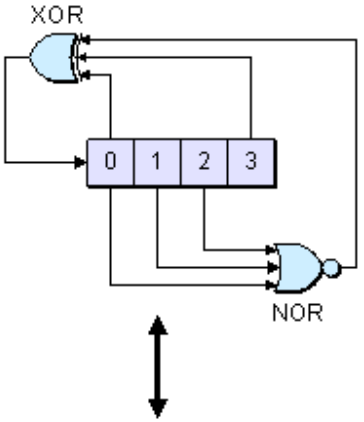
- For the value where all the bits are 0 to appear,
  - the preceding value must have comprised a logic 1 in the MSB and logic Os in the remaining bit positions.
- In an unmodified LFSR,
  - the next clock would result in a logic 1 in the LSB and logic Os in the remaining bit positions.
- However, in the modified LFSR,
  - the output from the NOR is a logic 0 for every case except the value preceding the one where all the bits are 0.



Clock Edge	q0	q1	q2	q3	
—	0	0	0	1	
1	0	0	0	0	← All 0's
2	1	0	0	0	
3	1	1	0	0	
4	1	1	1	0	
5	1	1	1	1	
6	0	1	1	1	
7	1	0	1	1	
8	0	1	0	1	2^4=16 Values
9	1	0	1	0	
10	1	1	0	1	
11	0	1	1	0	
12	0	0	1	1	
13	1	0	0	1	
14	0	1	0	0	
15	0	0	1	0	
16	0	0	0	1	

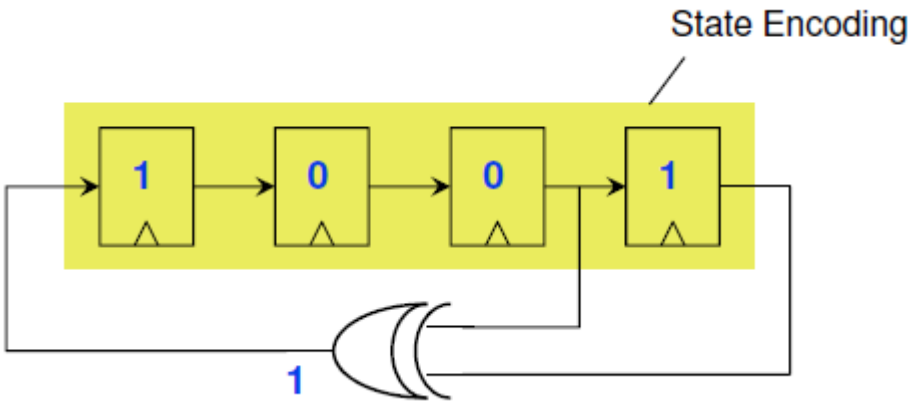
Modifying LFSRs to Sequence 2^n Values

- These two values force the NOR's output to a logic 1 which inverts the usual output from the XOR.
- This in turn causes the sequence to first enter the all Os value and then resume its normal course.
- In the case of LFSRs with XNOR feedback paths, the NOR can be replaced with an AND which causes the sequence to cycle through the value where all the bits are 1.



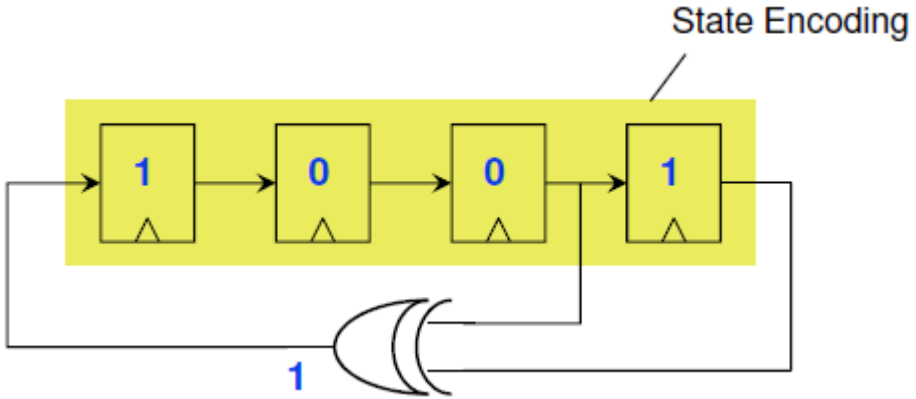
Clock	q0	q1	q2	q3	
--	0	0	0	1	Initial value
1	0	0	0	0	All 0s
2	1	0	0	0	
3	1	1	0	0	
4	1	1	1	0	
5	1	1	1	1	
6	0	1	1	1	
7	1	0	1	1	
8	0	1	0	1	
9	1	0	1	0	
10	1	1	0	1	
11	0	1	1	0	
12	0	0	1	1	
13	1	0	0	1	
14	0	1	0	0	
15	0	0	1	0	$2^4 = 16$ values
16	0	0	0	1	
17	0	0	0	0	
18	1	0	0	0	
:	:	:	:	:	

Accessing the Previous State Values



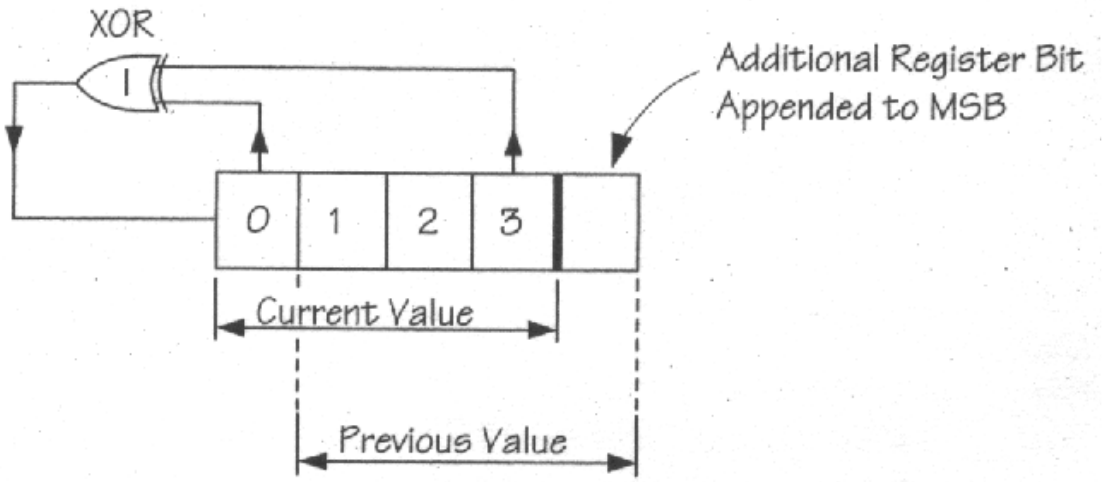
- In some applications, it is required to make use of a register's previous value.
- A trivial technique requires an additional set of shadow registers;
  - every time the counter is incremented, the current contents are first copied into the shadow registers.

Accessing the Previous Values



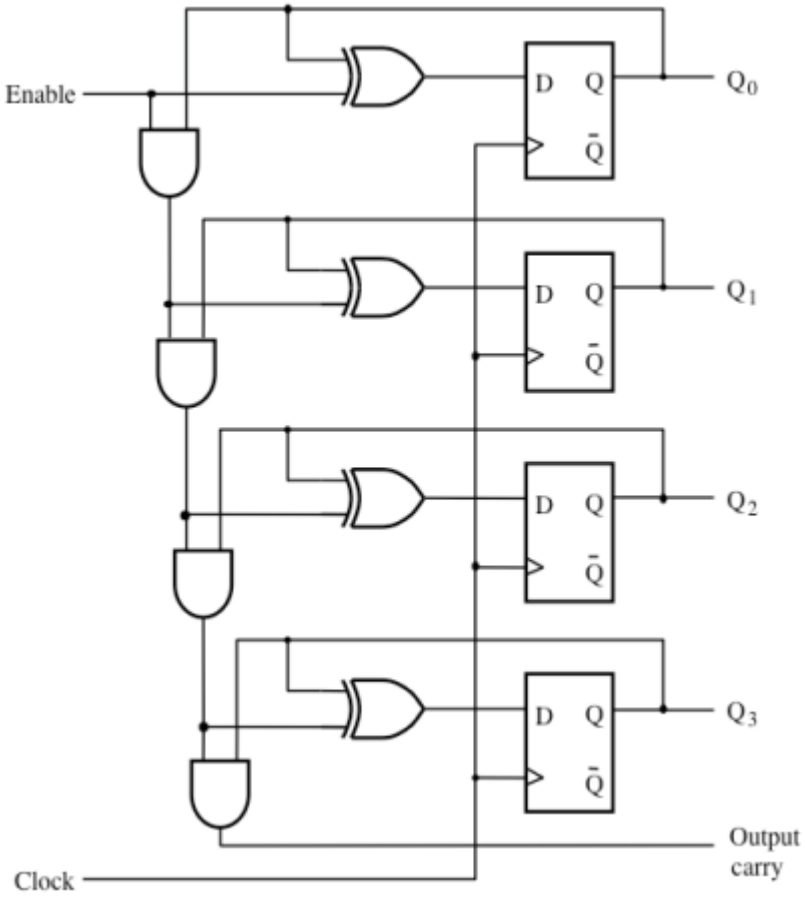
- Alternatively, a block of combinational logic can be used to decode the previous value from the current value.
- Unfortunately, both of these techniques involve a substantial overhead in terms of additional logic.

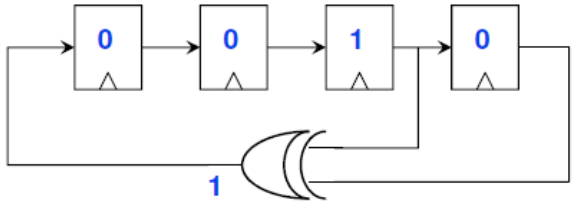
Accessing the Previous Values



- By comparison, LFSRs inherently remember their previous value;
  - the addition of a single register bit appended to the MSB.

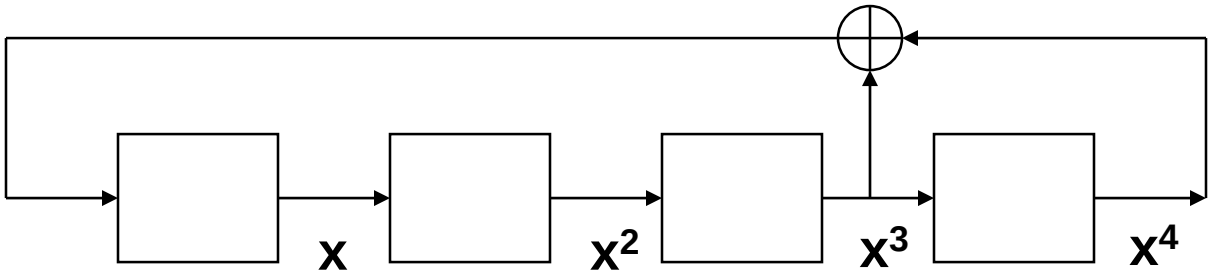
4-bit Counters



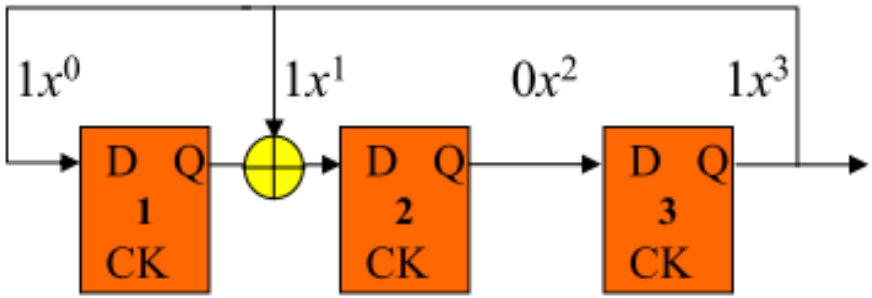


- The minimum clock period :
  - $T > T_{2\text{-input-xor}}$ + clock overhead
  - Very little latency
  - independent of n !
- This can be used as a fast counter, if the particular sequence of count values is not important.

Characteristic Polynomial of LFSR

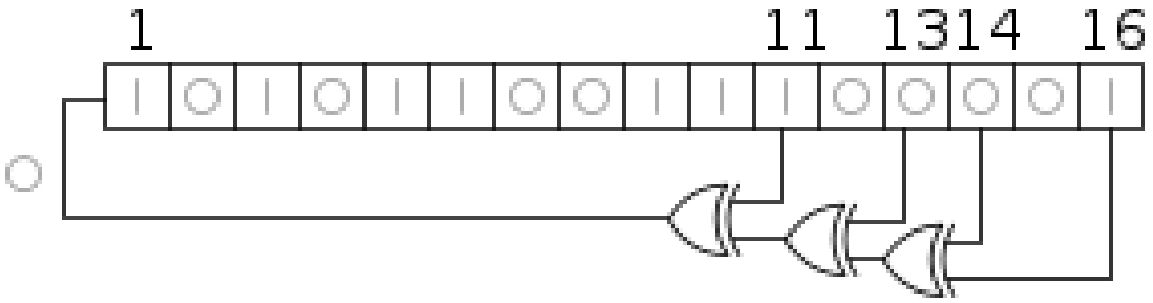


$P(x) = 1x^0 + 0x + 0x^2 + 1x^3 + 1x^4$



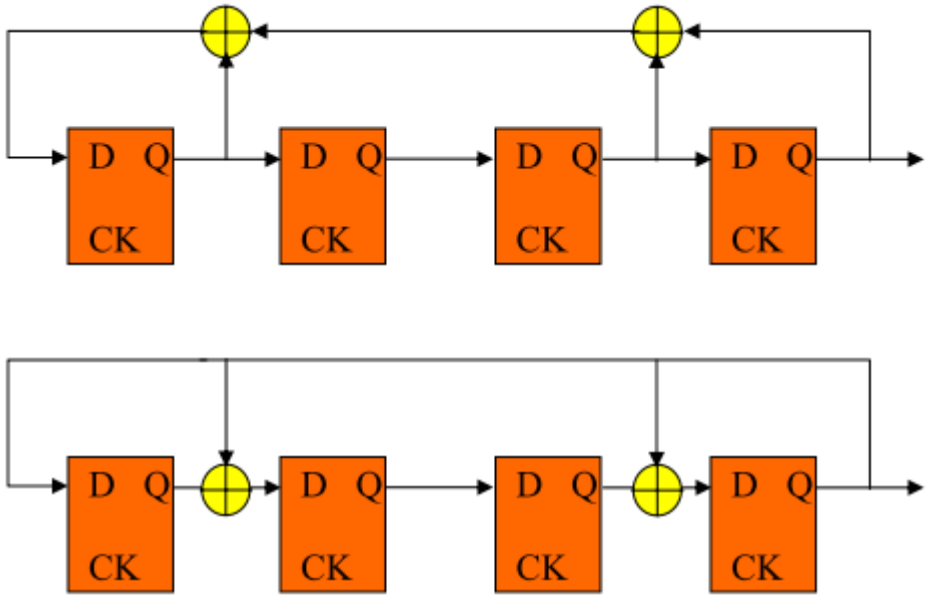
- The coefficients of the polynomial must be 1's or 0's.
  - Coefficient 1 signifies a connection
  - Coefficient 0 signifies no connection.

Example



$$x^{16} + x^{14} + x^{13} + x^{11} + 1.$$

Example



- $x^4 + x^3 + x + 1$ in both examples

Polynomial Theory

- A polynomial p(x) of degree n is primitive if
  - the remainders generated from the divisions of *all* polynomials with degree c ($c \leq 2^n-1$) by p(x) correspond to all possible non-zero polynomials of degree $\leq$ n.

- If we divide $x, x^2, x^3, x^4, x^5, x^6, x^7$ by polynomial $f(x)=x^3+x+1$
  - The remainders obtained are:
    - $x, x^2, x+1, x^2+x, x^2+x+1, x^2+1, 1$
- The division of all polynomials of degree 1 to 7 by f(x) results in
  - all 7 non-zero polynomials of degree $\leq$ 3.
  - f(x) is primitive.

Polynomial Theory

- A polynomial p(x) of degree n is primitive if
  - the remainders generated from the divisions of *all* polynomials with degree c ($c \leq 2^n-1$) by p(x) correspond to all possible non-zero polynomials of degree $\leq$ n.

- The division of $x, x^2, x^3, x^4, x^5, x^6, x^7$ by polynomial $g(x)=x^3+1$
- The remainders obtained are:
  - $1, x, x^2$, repeatedly
  - g(x) is not primitive

Theory for LFSRs: Galois Fields

- Theorem:
  - For any value n, there exists at least one feedback equation that makes the LFSR go through all 2^n-1 non-zero states before repeating.

- If (and only if) the polynomial is a primitive, then the LFSR is maximal.
- Example:
  - If an LFSR is implemented using $f(x)= x^3+x+1$, it will repeatedly generate 7 binary patterns in sequence.

LFSR

- The feedback connections needed to implement an LFSR can be derived directly from the chosen primitive polynomial.
- There can be more than one maximal tap sequence for a given LFSR length.

# of bits	Length of Loop	Taps
2	3	[0,1]
3	7	[0,2]
4	15	[0,3]
5	31	[1,4]
6	63	[0,5]
7	127	[0,6]
8	255	[1,2,3,7]
9	511	[3,8]
10	1023	[2,9]
11	2047	[1,10]
12	4095	[0,3,5,11]
13	8191	[0,2,3,12]
14	16383	[0,2,4,13]
15	32767	[0,14]
16	65535	[1,2,4,15]
17	131071	[2,16]
18	262143	[6,17]
19	524287	[0,1,4,18]
20	1,048,575	(2,19]
21	2,097,151	(1,20]
21	4,194,303	0,21]
23	8,388,607	[4,221
24	16,777,215	[0,2,3,23]
25	33,554,431	[2,24]
26	67,108,863	10,1,5,25]
27	134,217,727	[0,1,4,26]
28	268,435,455	[2,27]
29	536,870,911	[1,28]
30	1,073,741,823	[0,3,5,29]
31	2,147,483,647	[2,30]
32	4,294,967,295	[1,5,6,31]

LFSR

- Once one maximal tap sequence has been found, another automatically follows.
- Property:
  - If the tap sequence, in an n-bit LFSR, is [n,A,B,C], then the corresponding 'mirror' sequence is [n,n-A,n-B,n-C].
- Example:
  - The tap sequence [32,3,2,1] has as its counterpart [32,29,30,31].
  - Both give a maximal sequence.

LFSR

- In the SystemVerilog model, the feedback connections for LFSRs with 1 to 36 stages are defined in an **initial** block.
- The model is only defined for the range 1 to 36 (with a default value of 8).
- Any attempt to use this model for a larger LFSR would result in an invalid model.
- The **initial** block is evaluated once, at elaboration time, and the resulting value is used to configure the model.

```
module lfsr #(parameter N = 4)
(output logic [N-1:0] q,
input logic clock, n_set);
logic feedback;
int i;
logic [N-1:0] taps;
initial
begin
taps = '0;
case (N)
2: taps |= (1'b1 << 1);
3: taps |= (1'b1 << 1);
4: taps |= (1'b1 << 1);
5: taps |= (1'b1 << 2);
6: taps |= (1'b1 << 1);
7: taps |= (1'b1 << 1);
8: taps |= ((1'b1 << 6) | (1'b1 << 5)
| (1'b1 << 1));
9: taps |= (1'b1 << 4);
10: taps |= (1'b1 << 3);
11: taps |= (1'b1 << 2);
12: taps |= ((1'b1 << 7) | (1'b1 << 4)
```

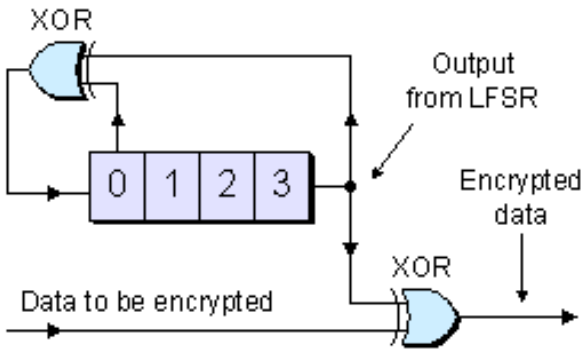
```
module lfsr #(parameter N = 4)
(output logic [N-1:0] q,
input logic clock, n_set);
logic feedback;
int i;
logic [N-1:0] taps;
initial
begin
taps = '0;
case (N)
2: taps |= (1'b1 << 1);
3: taps |= (1'b1 << 1);
4: taps |= (1'b1 << 1);
5: taps |= (1'b1 << 2);
6: taps |= (1'b1 << 1);
7: taps |= (1'b1 << 1);
8: taps |= ((1'b1 << 6) | (1'b1 << 5)
| (1'b1 << 1));
9: taps |= (1'b1 << 4);
10: taps |= (1'b1 << 3);
11: taps |= (1'b1 << 2);
12: taps |= ((1'b1 << 7) | (1'b1 << 4)
| (1'b1 << 3));
13: taps |= ((1'b1 << 4) | (1'b1 << 3)
| (1'b1 << 1));
14: taps |= ((1'b1 << 12) | (1'b1 << 11)
| (1'b1 << 1));
15: taps |= (1'b1 << 1);
16: taps |= ((1'b1 << 5) | (1'b1 << 3)
| (1'b1 << 2));
17: taps |= ((1'b1 << 3));
18: taps |= ((1'b1 << 7));
19: taps |= ((1'b1 << 6) | (1'b1 << 5)
| (1'b1 << 1));
20: taps |= (1'b1 << 3);
21: taps |= (1'b1 << 2);
22: taps |= (1'b1 << 1);
23: taps |= (1'b1 << 5);
24: taps |= ((1'b1 << 4) | (1'b1 << 3)
| (1'b1 << 1));
25: taps |= (1'b1 << 3);
26: taps |= ((1'b1 << 8) | (1'b1 << 7)
| (1'b1 << 1));
27: taps |= ((1'b1 << 8) | (1'b1 << 7)
| (1'b1 << 1));
28: taps |= (1'b1 << 3);
29: taps |= (1'b1 << 2);
30: taps |= ((1'b1 << 16) | (1'b1 << 15)
| (1'b1 << 1));
31: taps |= (1'b1 << 3);
32: taps |= ((1'b1 << 28) | (1'b1 << 27)
| (1'b1 << 1));
33: taps |= (1'b1 << 13);
34: taps |= ((1'b1 << 15) | (1'b1 << 14)
| (1'b1 << 1));
35: taps |= (1'b1 << 2);
36: taps |= (1'b1 << 11);
endcase
end
```

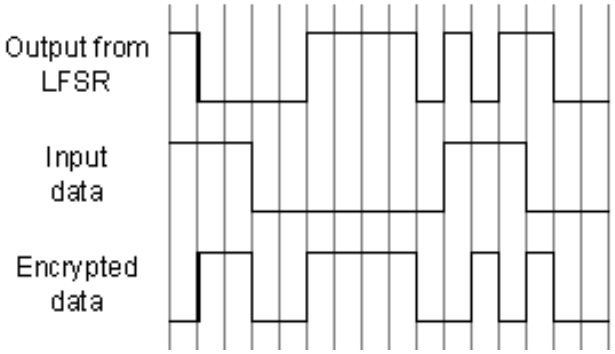
Linear Feedback Shift Register

```
always_ff @(posedge clock, negedge n_set)
    if (~n_set)
        q <= '1;
    else
        q <= {feedback, q[N-1:1]};
always_comb
    begin
        feedback = q[0];
        for (i = 1; i <= N - 1; i++)
        if (taps[i])
        feedback ^= q[i];
    end
endmodule
```

- The model defines up to three feedback connections – it is assumed that bit 0 is always used.
- The positions corresponding to the feedback connections are set to 1, using a shift operator and OR-ing theses values with the initial, all 0s value, using the assign and OR operator, |=.
- To construct the feedback connection for a particular size of LFSR, the stages of the LFSR referenced in the taps vector are XORed together using a **for** loop.

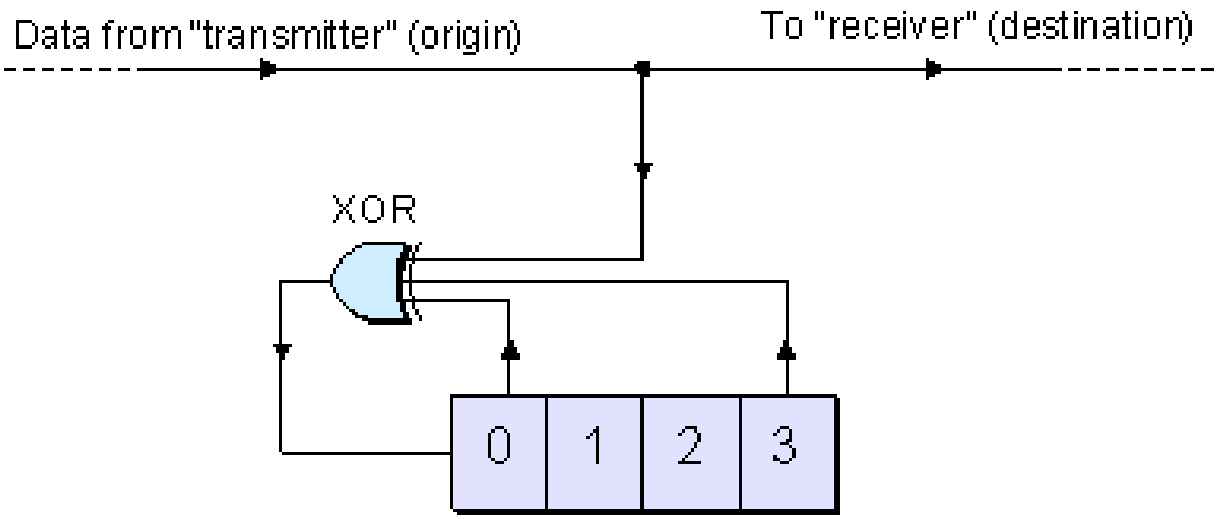
Encryption and Decryption Applications





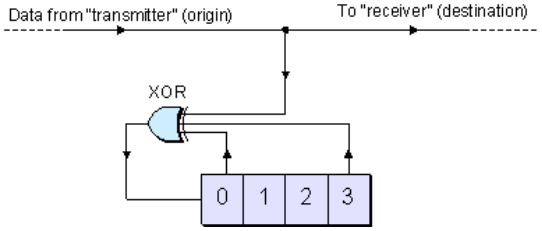
- The unusual sequence of values generated by an LFSR can be gainfully employed in the encryption (scrambling) and decryption (unscrambling) of data.
- A stream of data bits can be encrypted by XOR-ing them with the output from an LFSR.
- The stream of encrypted data bits seen by a receiver can be decrypted by XOR-ing them with the output of an identical LFSR.
- This is a trivial form of encryption that's not very secure, but it's "cheap-and-cheerful" and may be useful in certain applications.

Cyclic Redundancy Check (CRC)



- A traditional application for LFSRs is in cyclic redundancy check (CRC) calculations, which can be used to detect errors in data communications.
- The stream of data bits being transmitted is used to modify the values fed back into an LFSR.

Cyclic Redundancy Check (CRC)



- The final CRC value stored in the LFSR is known as a *checksum*, and is dependent on every bit in the data stream.
- After all of the data bits have been transmitted, the transmitter sends its checksum value to the receiver.
- The receiver contains an identical CRC calculator and generates its own checksum value from the incoming data.
- Once all of the data bits have arrived, the receiver compares its internally generated checksum value with the checksum sent by the transmitter to determine whether any corruption occurred during the course of the transmission.
- This form of error detection is very efficient in terms of the small number of bits that have to be transmitted in addition to the data.

References

- Application-Specific Integrated Circuits Addison-Wesley VLSI Design Series ISBN: 0-201-50022-1 TK7874.6.S63 Michael John Sebastian Smith Book on Web .
- HDL Chip Design : A Practical Guide for Designing, Synthesizing & Simulating ASICs & FPGAs Using VHDL or Verilog, Douglas J. Smith, 1998. Doone Pubns; ISBN: 0965193438
- Digital System Design with SystemVerilog by Mark Zwolinski. Prentice Hall, 2009.
- Language Reference Manual (LRM) – IEEE 1800-2005
- Bebop to the Boolean Boogie: An Unconventional Guide to Electronics.