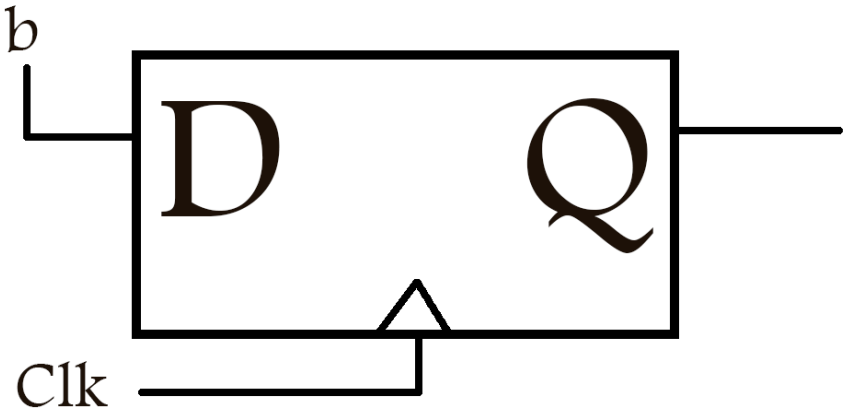
Security 1 - Lab 1

LFSR

Register (D flip-flop)

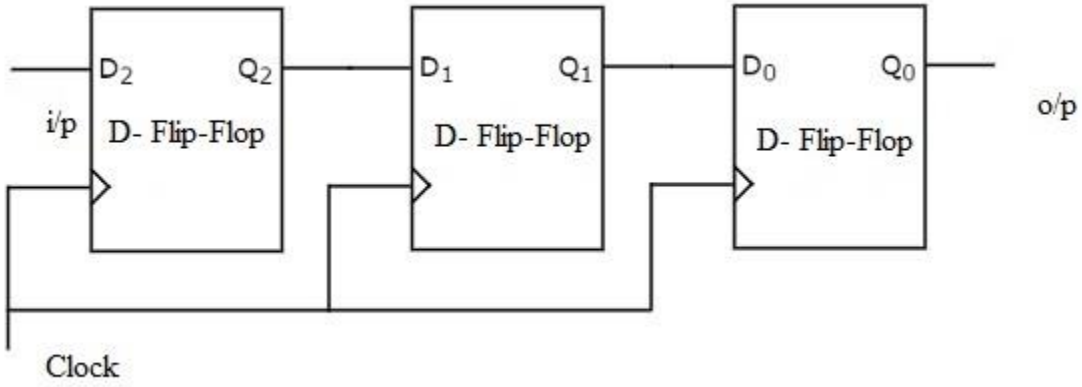
- Outputs Q when activated (Clock's 1st trigger).
- Changes Q to the input coming through 'D' at the end of activation (Clock's 2nd trigger).
- Basic circuit has multiple uses when combined with other circuits.
- One of the basic functionality is storing a single bit.



Shift Registers

- Shift Registers are a sequence of registers.
- Each register in the sequence is connected to only one register from the output side and only one from input side (Except the 1st and last one)
- An example use of such register can be delaying and storing a sequence of inputs.
- Starting from the input side of the register (usually left in the diagram) the input is stored in the first register on the 1st clock trigger.
- Then the 1st input is passed to the 2nd on the

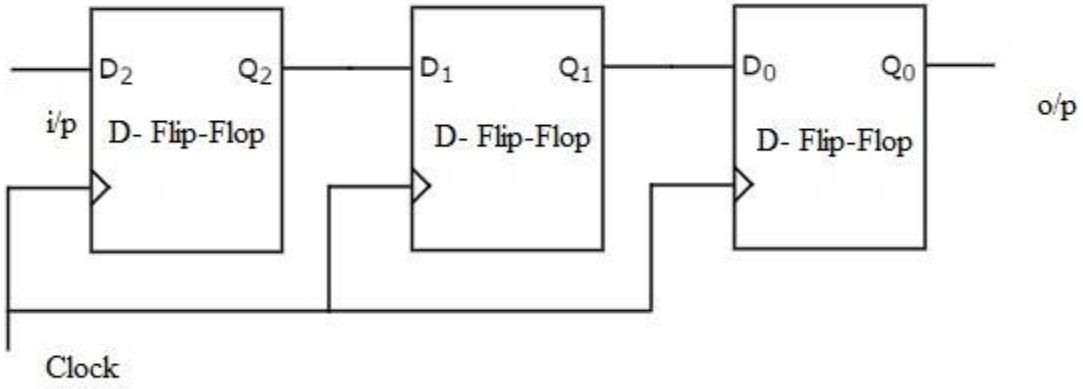
2nd trigger and the 2nd input is passed to 1st register.



Shift Registers

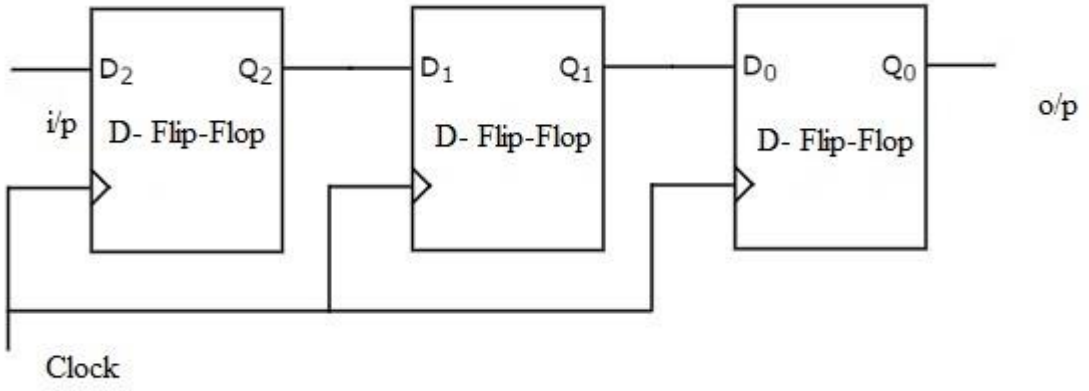
- The input keeps on being passed or “Shifted” onto the next register until it is outputted at the last one.
- The output of the last register is the output we take into consideration, everything in between is an intermediate stage.
- The value outputted at time ‘i’ is S(i). Hence if there are ‘n’ registers, we will need to wait ‘n’ clocks to get the first input.
- The first input will come out at the ‘n’th clock,

so it will be s(n), or s(n+0), the 2nd input s(n+1), etc…



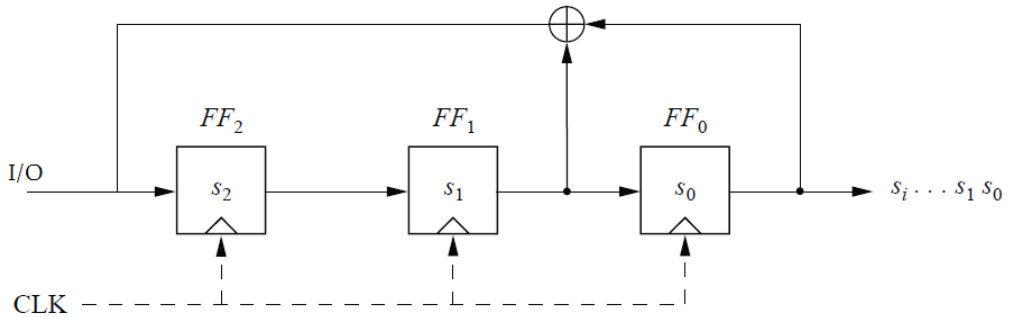
Shift Registers

- So what are the values s(0) to s(n-1)?
- These are the values that were in the registers when we started inputting the stream.
- The initial states of the registers are given the names FF_0, FF_1, FF_2… FF_{n-1}, and these are the values outputted for s(0) to s(n-1) in this basic demonstrated design (note that it can be changed according to design choice).



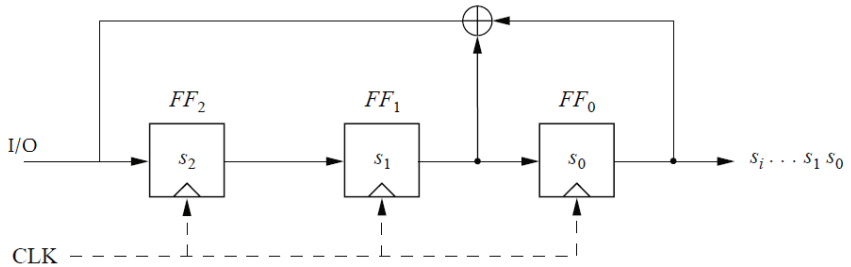
Linear Feedback shift registers (LFSR)

- The network shown above has the following output function
  s(i) = s(i-2) ⊕ s(i-3) // For 'i' >= 3 and s(i) = FF(i) for 'i' < 3 & 'i' >= 0
  (or mainly $s_{i+3} = s_{i+1} \oplus s_i$ // The use of XOR is equivalent to adding and using mod 2: $s_{i+3} = s_{i+1} + s_i$ Mod2)
- Where 's' is the output function for our cipher key. And 'FF' are the initial states.
- Note that in this diagram, the input for the left most register is taken as a linear combination of the others registers' output.



Linear Feedback shift registers (LFSR)

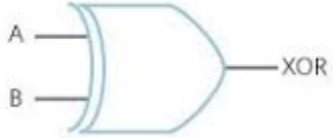
- Using the initial states of $FF_2 = 1$, $FF_1 = 0$, $FF_0 = 0$, we get a repeating pattern output of 0010111 0010111 0010111…
- This register has a period length of 7, this can be different if we used different starting values for the registers. And it can be different at the beginning then it starts to repeat the same pattern stated above (it converges to the same pattern).
- Luckily, we have at least two options to add to the register.
- We can increase the number of registers, we can change what gets added when giving the feed back to the first register.



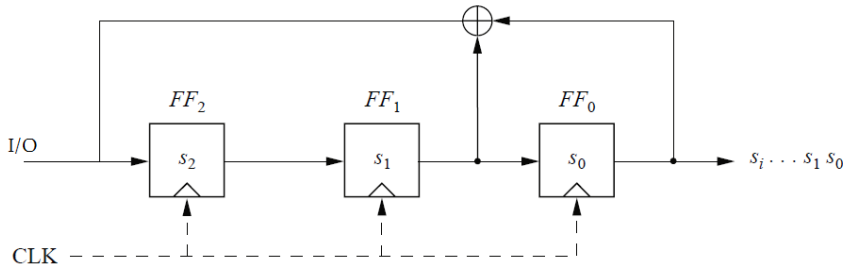
Linear Feedback shift registers (LFSR)

- The maximum sequence for an LFSR of degree m is 2^m-1.
- Since m-bit state vector can only have 2^m-1 nonzero states, the max sequence length is 2^m-1.
- The all zero state must be excluded. If an LFSR assumes this state, it will get stuck in it (Check the design of LFSR and figure out why).
- Only certain configurations can get us a sequence of max length.

$X = A \oplus B$

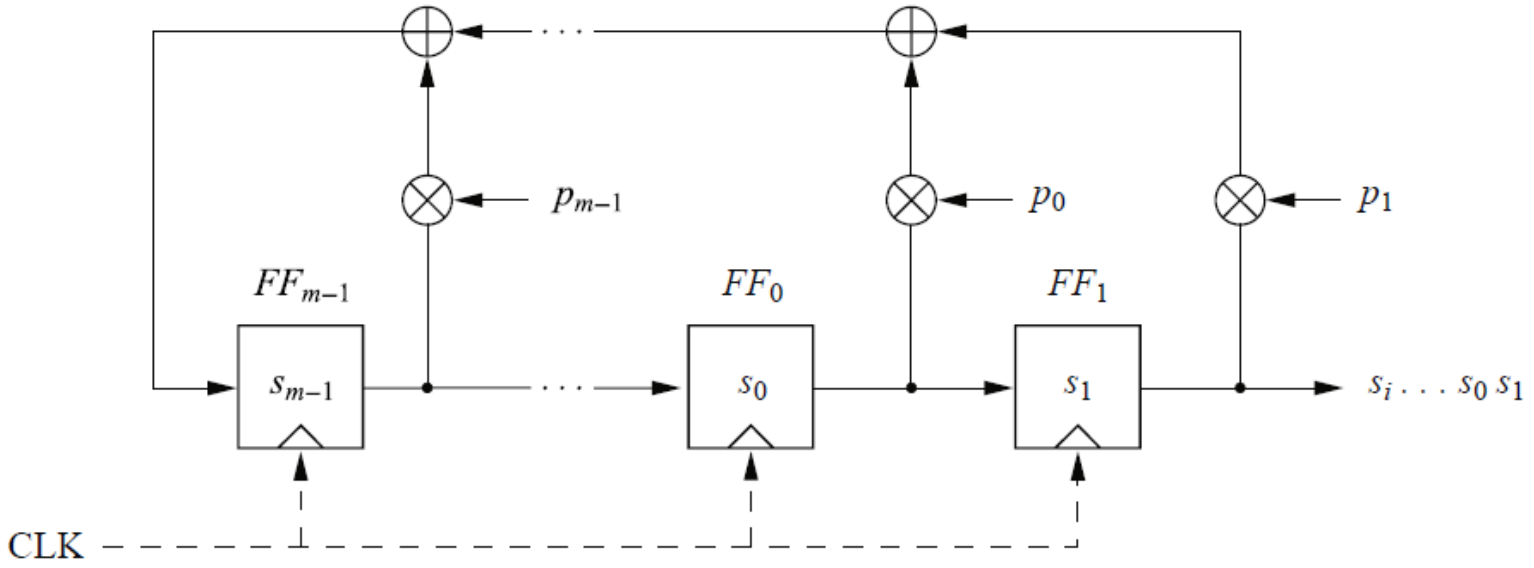


A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0



Linear Feedback shift registers (LFSR)

- A generalization of how it can look.

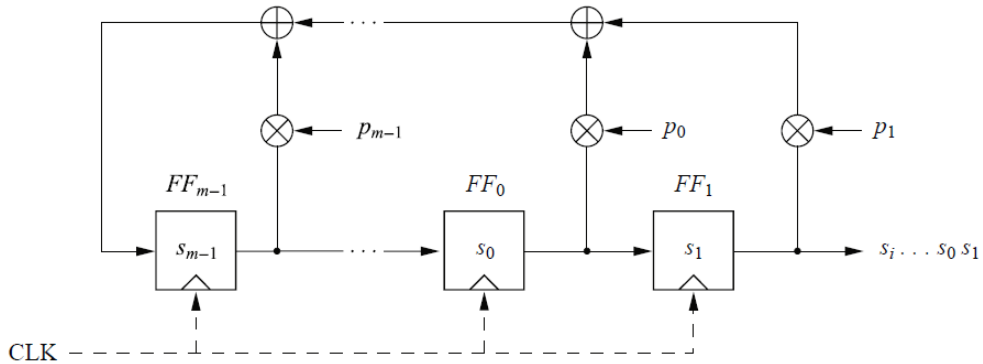


Linear Feedback shift registers (LFSR)

- In the diagram shown in the previous slide. We have the option to control what can be added through the gates 'p_i'. The value of the gates p(i) is ANDed with whatever is inputted into them.
- After every gate, the results are all XORed and connected back to the input of the 1st / left most register.
- How this design works depends on the XOR function. First any of 0/1 XORed with 0 gives us the 0/1 (same input taken). So when p(i) is set to 0, it will always output 0 and it won't affect the result of the XOR.
- If p(i) is 1, then 1 AND anything (1/0) is the anything

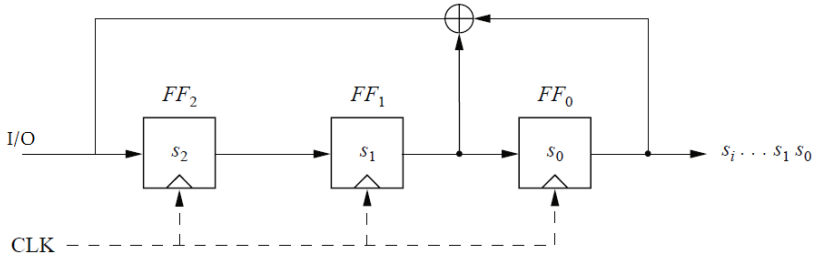
(1/0) and at that state the XOR will change according to

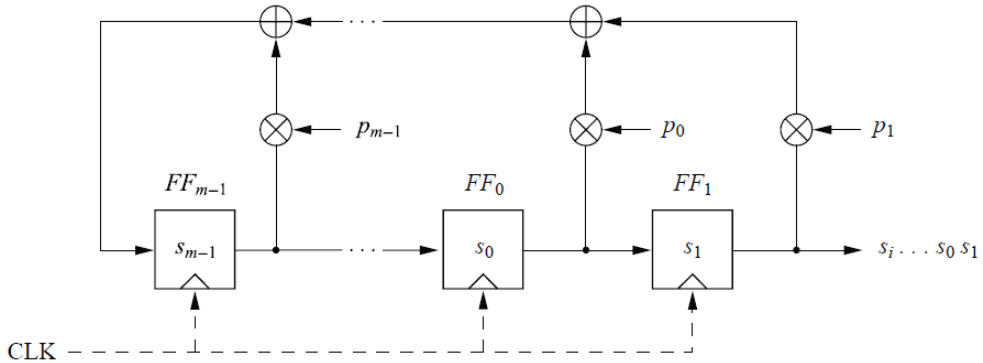
it's input.



Linear Feedback shift registers (LFSR)

- Now the equations for the ciphers bits outputted becomes:
  $s(i+m) = s_{i+m-1} * p_{i+m-1} + s_{i+m-2} * p_{i+m-2} + \ldots + s_i * p_0 \mod 2$
- LSFR are often specified as polynomials of the form:
  $P(x) = x^m + p_{m-1}x^{m-1} + \ldots p_1 x + p_0$
- The 1st X tells the number of registers, and the rest tells us the included ones
- The polynomial in the given example (before previous diagram) is:
  $P(x) = X^3 + X + 1$ (3 registers last two included)





LFSR: Known-plaintext attack

- The linearity in the cipher makes it easy to break under certain conditions.
- If the attacker has a plaintext – ciphertext pair, he will now know what s(i) is. For every s(i) the attacker knows he can make up a linear equation with a known value of s(i).
- Given enough linear equations he can solve them to know p(x). And knowing p(x) is the key. Note that further knowing the size of the registers ‘m’ further helps the attacker.

Questions

2.1. The stream cipher described in Definition 2.1.1 can easily be generalized to work in alphabets other than the binary one. For manual encryption, an especially useful one is a stream cipher that operates on letters.

1. Develop a scheme which operates with the letters A, B,. . ., Z, represented by the numbers 0,1,. . .,25. What does the key (stream) look like? What are the encryption and decryption functions?
2. Decrypt the following cipher text:
   `bsaspp kkuosp`
   which was encrypted using the key:
   `rsidpy dkawoa`

Questions

2.5. We will now analyze a pseudorandom number sequence generated by a LFSR characterized by $(c_2 = 1, c_1 = 0, c_0 = 1)$.

1. What is the sequence generated from the initialization vector $(s_2 = 1, s_1 = 0, s_0 = 0)$?

Questions

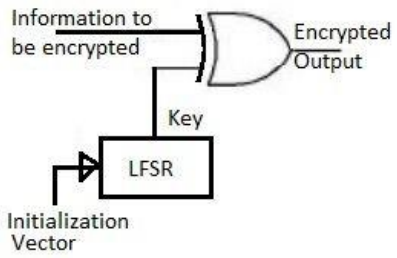
2.8. In this problem we will study LFSRs in somewhat more detail. LFSRs come in three flavors:

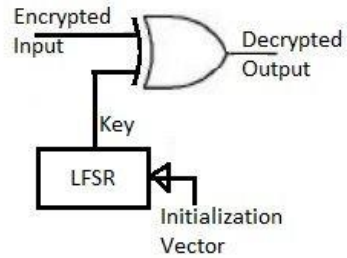
- LFSRs which generate a maximum-length sequence. These LFSRs are based on *primitive polynomials*.
- LFSRs which do not generate a maximum-length sequence but whose sequence length is independent of the initial value of the register. These LFSRs are based on *irreducible polynomials* that are not primitive. Note that all primitive polynomials are also irreducible.
- LFSRs which do not generate a maximum-length sequence and whose sequence length depends on the initial values of the register. These LFSRs are based on *reducible polynomials*.

We will study examples in the following. Determine *all* sequences generated by

$$x^4 + x^2 + 1$$

Draw the corresponding LFSR for each of the three polynomials. Which of the polynomials is primitive, which is only irreducible, and which one is reducible? Note that the lengths of all sequences generated by each of the LFSRs should add up to $2^m - 1$.





Assignment

- Use LFSR to encrypt and decrypt a message with the following requirements for the LFSR :
- The LFSR will be of size 9 bits (fixed).
- The initial state (1/0) and the values of the gates P_i are to be read from a file (preferred to be the same file). Along with the number of bits that will be used for warm up 'X' described below.
- Warm up Phase: Discard the first X bits outputted by the LFSR, but display them on console (Show them as output, but don't use them in encryption/decryption).
- After that starting from the (X+1) bit, start using the output of the LFSR to encrypt the input plain text and output the result of the encryption. Then decrypt the ciphered text and output the result of the decryption.
- Will be submitted on blackboard by max 6th of Nov 2021, and will be discussed on the next lab.