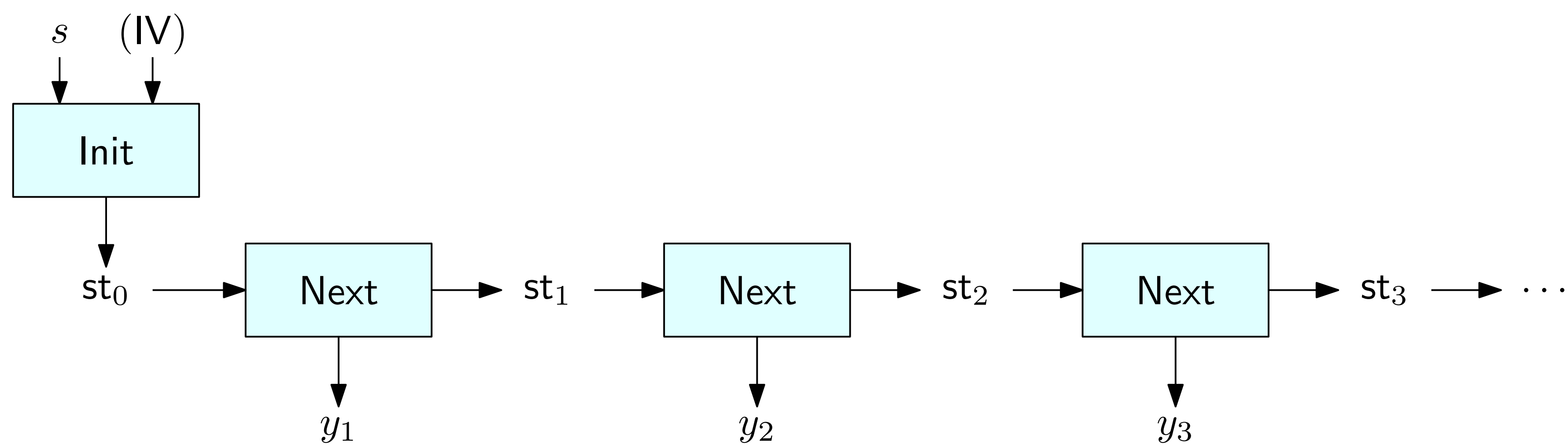
Stream ciphers (reminder)

A stream cipher is a pair of deterministic polynomial-time algorithms

- **Init:** takes a n-bit seed s, and possibly a n-bit *initialization vector* (IV), and outputs a *state* st
- **Next:** takes a state st and outputs a bit y and a new (updated) state st′

Idea: we can generate as many random bits as desired, by repeatedly calling Next



* In practice, **Next** can output multiple bits at once (e.g., a byte)

Stream ciphers (reminder)

If the stream cipher does not support IVs, then it should behave like a PRG

- For a key chosen u.a.r., its output should be indistinguishable (to poly-time adversaries) from a uniform stream of random bits chosen independently at random (as long as the output length is polynomial)

If the stream cipher does support IVs, then the stream cipher should behave like a PRF

- For any key (chosen u.a.r.) the output streams generated from multiple IVs (chosen u.a.r.) should be indistinguishable (to poly-time adversaries) from multiple streams of random bits, where each bit is chosen u.a.r.

- This must still be true even if the adversary is given the IVs!

Stream ciphers (reminder)

- We don't know if (secure) stream ciphers exist (we don't know if PRGs / PRFs exist)
- In practice we have some candidate stream cipher constructions that are conjectured to be secure
- These construction have withstood years of public scrutiny and attempted cryptanalysis

- Some popular practical constructions of stream ciphers:
  - Trivium: optimized for hardware
  - RC4 (insecure): optimized for software
  - ChaCha20: replacement of RC4

Trivium

- Stream cipher selected as part of the eSTREAM portfolio

European project to "identify new stream ciphers suitable for widespread adoption"

Trivium

- Stream cipher selected as part of the eSTREAM portfolio

> European project to "identify new stream ciphers suitable for widespread adoption"

- Designed to be easy to implement in hardware
- "designed as an exercise in exploring how far a stream cipher can be simplified without sacrificing its security, speed or flexibility"

Trivium

- Stream cipher selected as part of the eSTREAM portfolio

> European project to "identify new stream ciphers suitable for widespread adoption"

- Designed to be easy to implement in hardware

- "designed as an exercise in exploring how far a stream cipher can be simplified without sacrificing its security, speed or flexibility"

- No cryptanalytic attacks better than exhaustive search are currently known

  Although some attacks against "reduced" versions Trivium are known

Trivium

- Stream cipher selected as part of the eSTREAM portfolio

> European project to "identify new stream ciphers suitable for widespread adoption"

- Designed to be easy to implement in hardware
- "designed as an exercise in exploring how far a stream cipher can be simplified without sacrificing its security, speed or flexibility"
- No cryptanalytic attacks better than exhaustive search are currently known

  Although some attacks against "reduced" versions Trivium are known

- Based on feedback shift registers (FSR)

Trivium

- Stream cipher selected as part of the eSTREAM portfolio

> European project to "identify new stream ciphers suitable for widespread adoption"

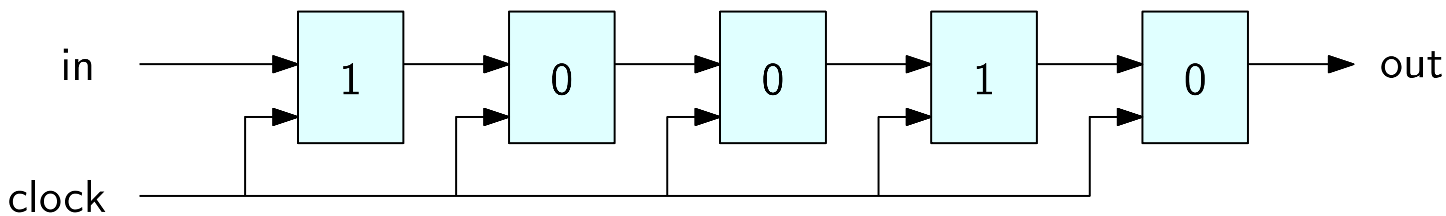
- Designed to be easy to implement in hardware
- "designed as an exercise in exploring how far a stream cipher can be simplified without sacrificing its security, speed or flexibility"
- No cryptanalytic attacks better than exhaustive search are currently known

  Although some attacks against "reduced" versions Trivium are known

- Based on feedback shift registers (FSR)

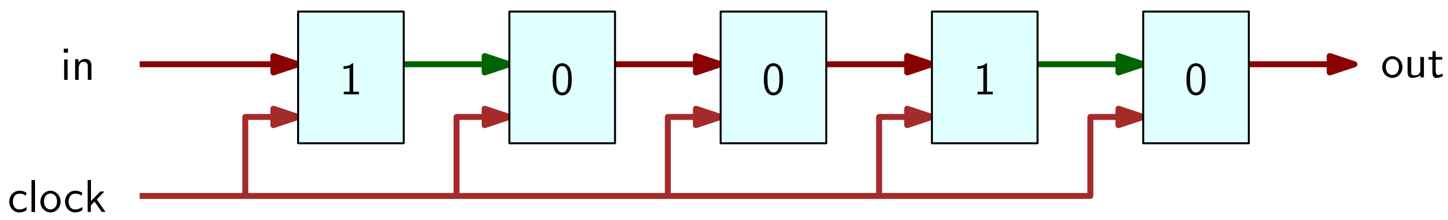
Shift Registers

- Shift register with n bits



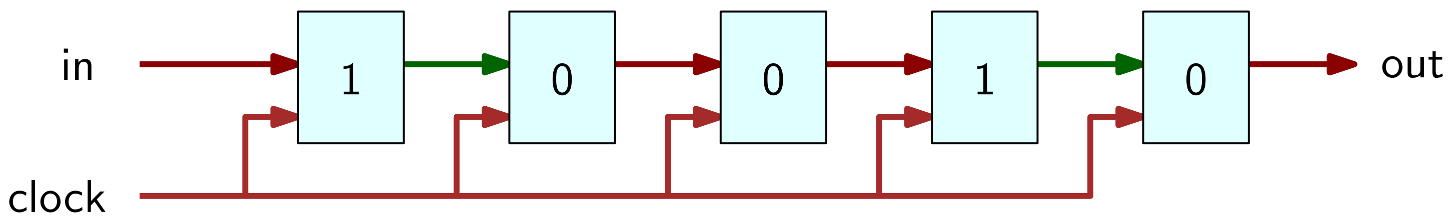
Shift Registers

- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



Shift Registers

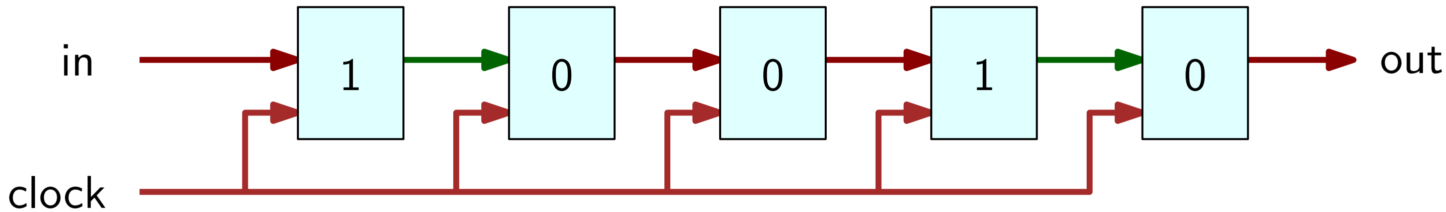
- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



- The output is the bit stored in the last register

Shift Registers

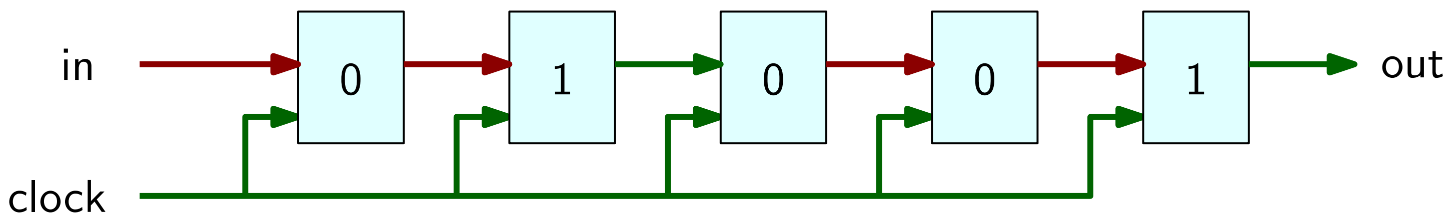
- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



- The output is the bit stored in the last register
- The leftmost bit is updated to the value of the “in” input line

Shift Registers

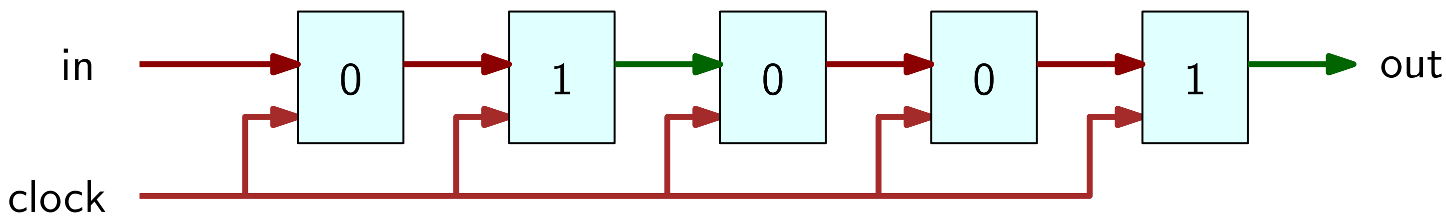
- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



- The output is the bit stored in the last register
- The leftmost bit is updated to the value of the “in” input line

Shift Registers

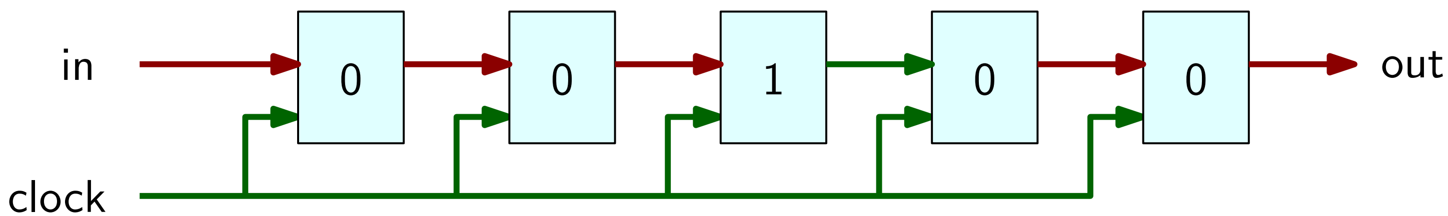
- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



- The output is the bit stored in the last register
- The leftmost bit is updated to the value of the "in" input line

Shift Registers

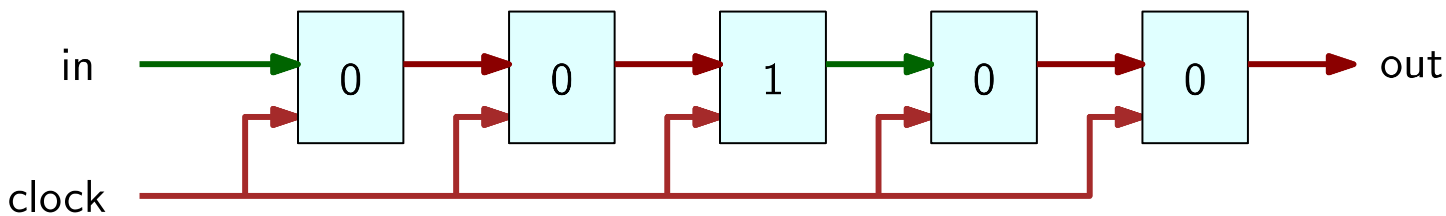
- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



- The output is the bit stored in the last register
- The leftmost bit is updated to the value of the “in” input line

Shift Registers

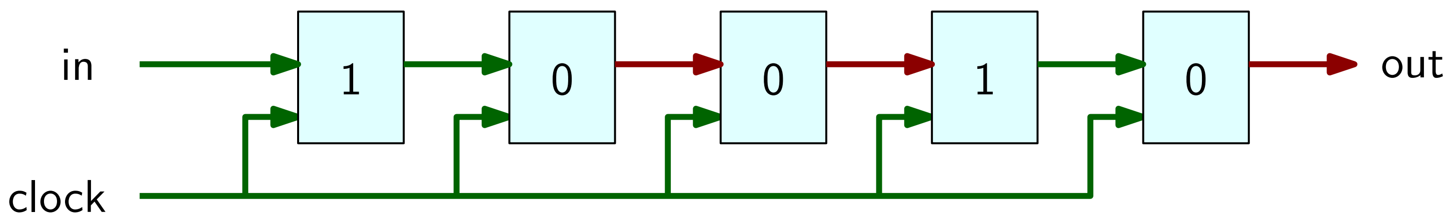
- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



- The output is the bit stored in the last register
- The leftmost bit is updated to the value of the “in” input line

Shift Registers

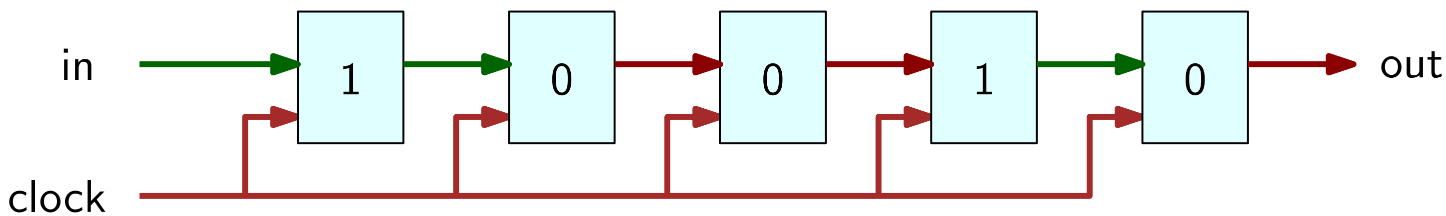
- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



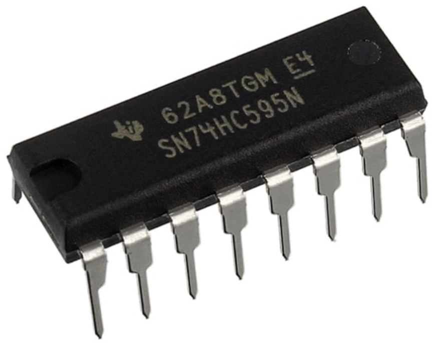
- The output is the bit stored in the last register
- The leftmost bit is updated to the value of the “in” input line

Shift Registers

- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick

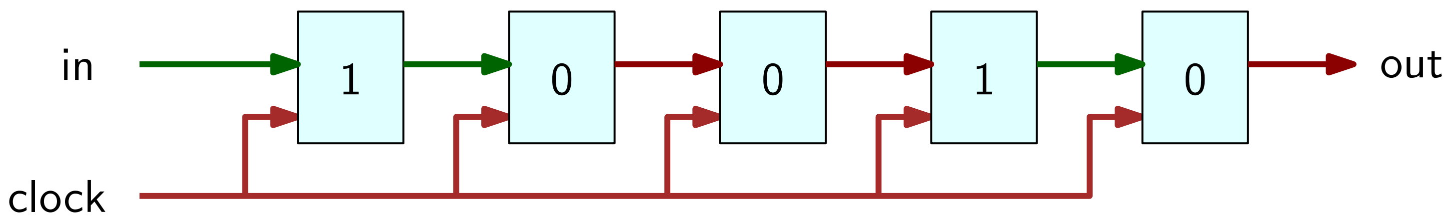


- The output is the bit stored in the last register
- The leftmost bit is updated to the value of the "in" input line



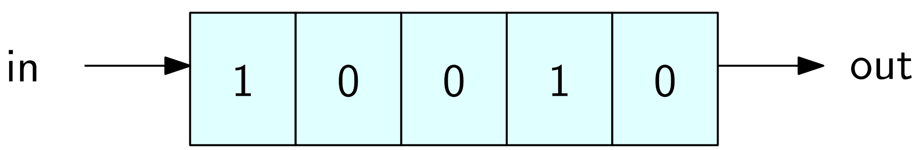
Shift Registers

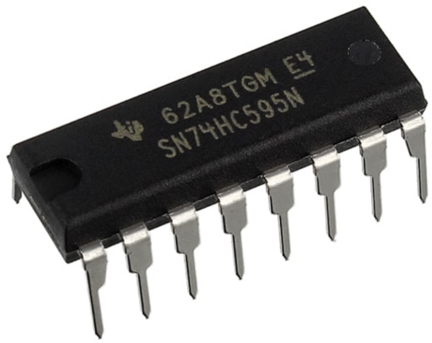
- Shift register with n bits
- The stored bits update (their values shift to the right) at each clock tick



- The output is the bit stored in the last register
- The leftmost bit is updated to the value of the "in" input line

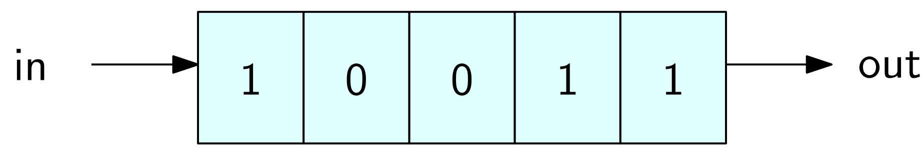
We use a simplified graphical depiction:





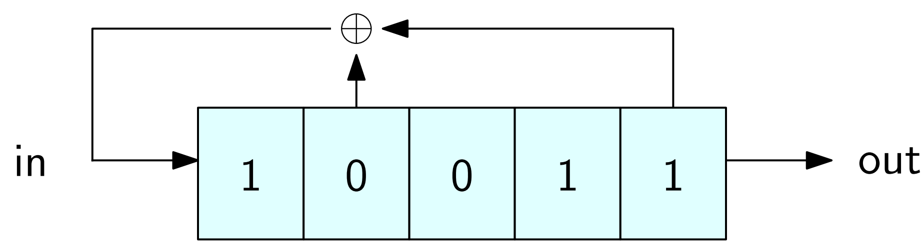
Linear Feedback Shift Registers (LFSR)

The value of the “in” line is the XOR of a subset of the bits in the register



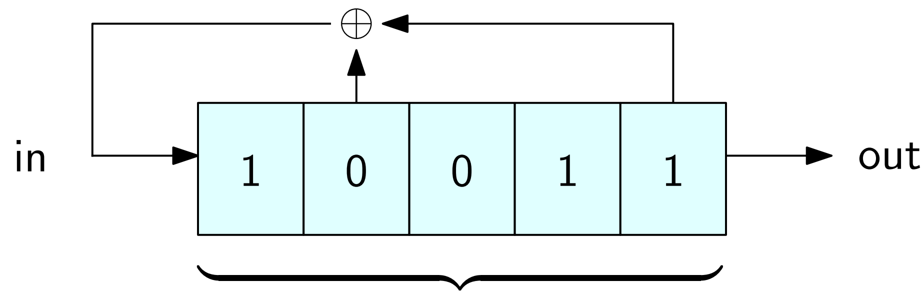
Linear Feedback Shift Registers (LFSR)

The value of the “in” line is the XOR of a subset of the bits in the register



Linear Feedback Shift Registers (LFSR)

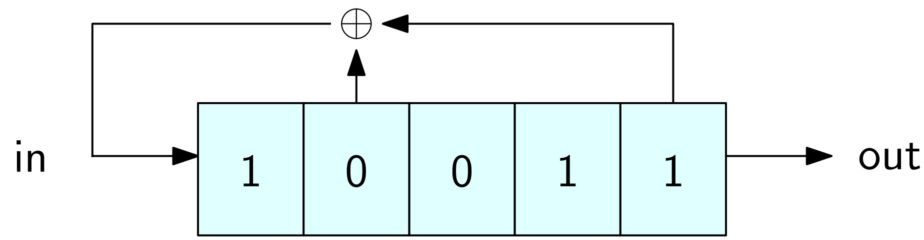
The value of the “in” line is the XOR of a subset of the bits in the register



the number of bits is called
the **degree** of the FSR

Linear Feedback Shift Registers (LFSR)

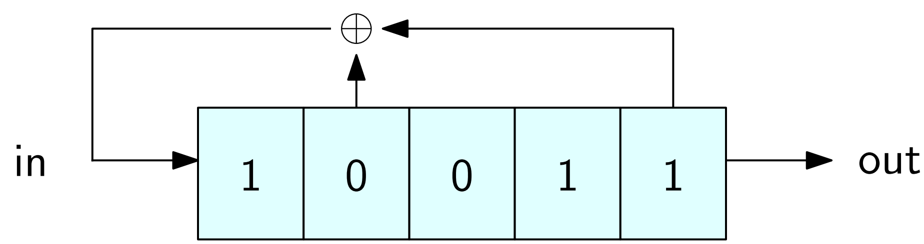
The value of the “in” line is the XOR of a subset of the bits in the register



The content of the register is called the **state** of the register

Linear Feedback Shift Registers (LFSR)

The value of the "in" line is the XOR of a subset of the bits in the register



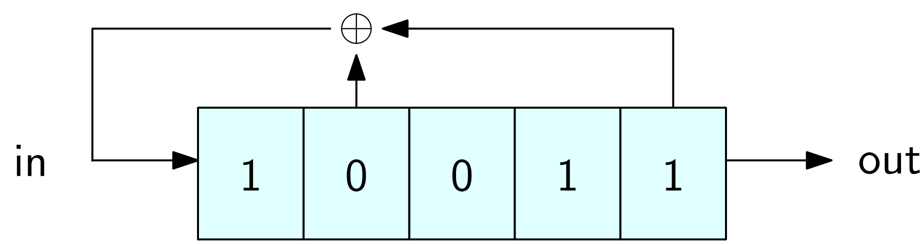
The content of the register is called the **state** of the register

At each clock tick:

- One bit is output
- The state is updated

Linear Feedback Shift Registers (LFSR)

The value of the "in" line is the XOR of a subset of the bits in the register



The content of the register is called the **state** of the register

At each clock tick:

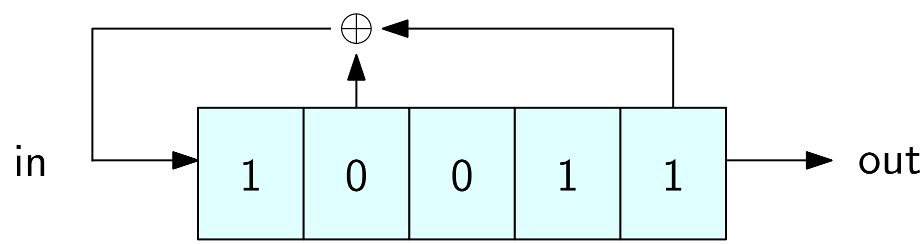
- One bit is output
- The state is updated

Sequence of states and output bits in the above example:

- States: 10011
- Outputs:

Linear Feedback Shift Registers (LFSR)

The value of the "in" line is the XOR of a subset of the bits in the register



The content of the register is called the **state** of the register

At each clock tick:

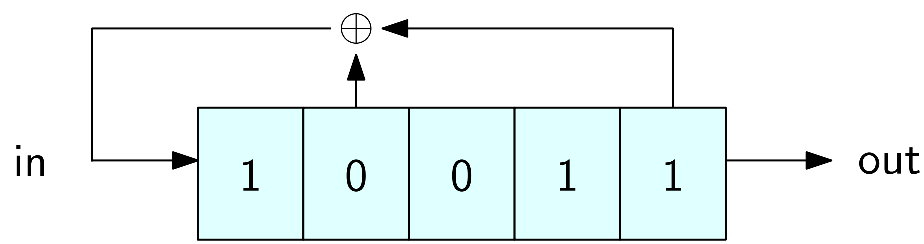
- One bit is output
- The state is updated

Sequence of states and output bits in the above example:

- States: 10011 $\rightarrow$ 11001
- Outputs: 1

Linear Feedback Shift Registers (LFSR)

The value of the “in” line is the XOR of a subset of the bits in the register



The content of the register is called the **state** of the register

At each clock tick:

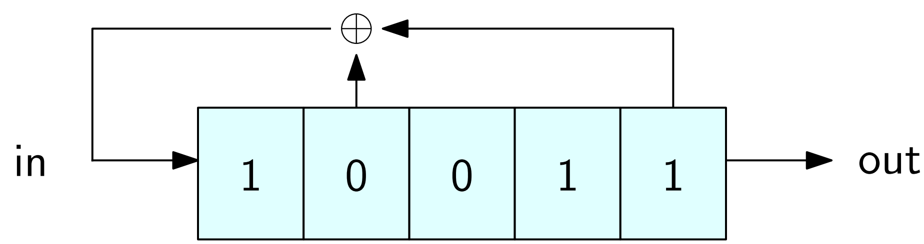
- One bit is output
- The state is updated

Sequence of states and output bits in the above example:

- States: 10011 $\rightarrow$ 11001 $\rightarrow$ 01100
- Outputs: 1 1

Linear Feedback Shift Registers (LFSR)

The value of the "in" line is the XOR of a subset of the bits in the register



The content of the register is called the **state** of the register

At each clock tick:

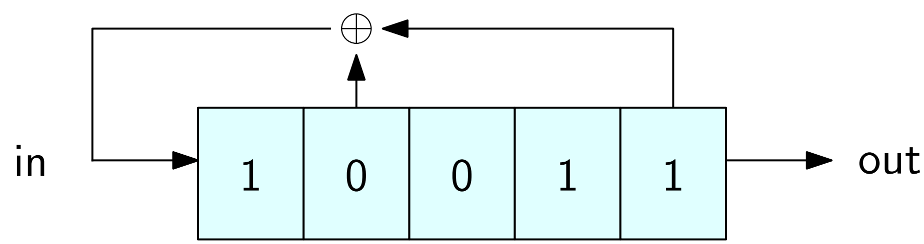
- One bit is output
- The state is updated

Sequence of states and output bits in the above example:

- States: 10011 $\rightarrow$ 11001 $\rightarrow$ 01100 $\rightarrow$ 10110
- Outputs: 1 1 0

Linear Feedback Shift Registers (LFSR)

The value of the "in" line is the XOR of a subset of the bits in the register



The content of the register is called the **state** of the register

At each clock tick:

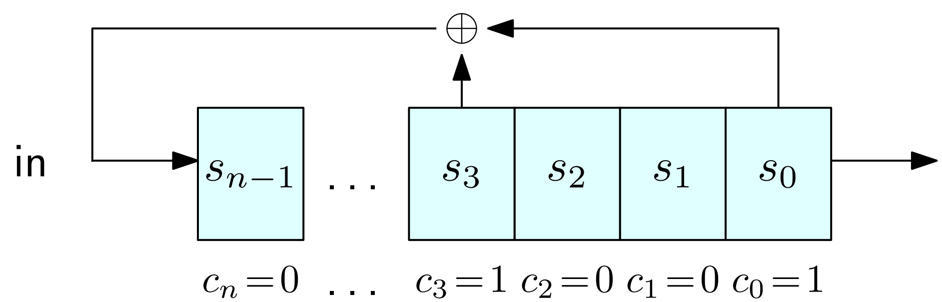
- One bit is output
- The state is updated

Sequence of states and output bits in the above example:

- States: $10011 \rightarrow 11001 \rightarrow 01100 \rightarrow 10110 \rightarrow 01011 \rightarrow \cdots$
- Outputs: $1 \quad 1 \quad 0 \quad 0$

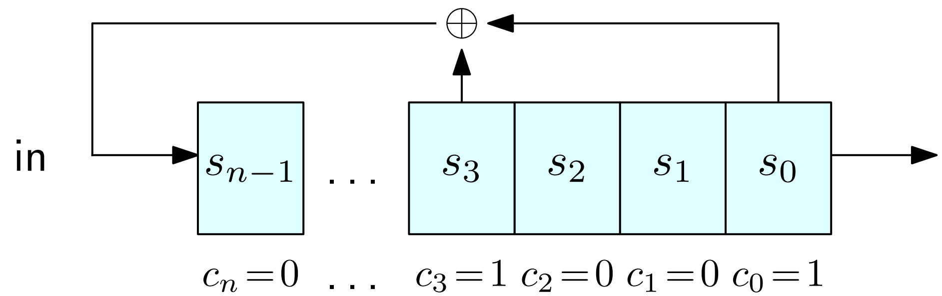
Linear Feedback Shift Registers (LFSR)

The subset of bits that are XOR-ed together can be described by n coefficients $c_0, c_1, \ldots, c_{n-1}$



Linear Feedback Shift Registers (LFSR)

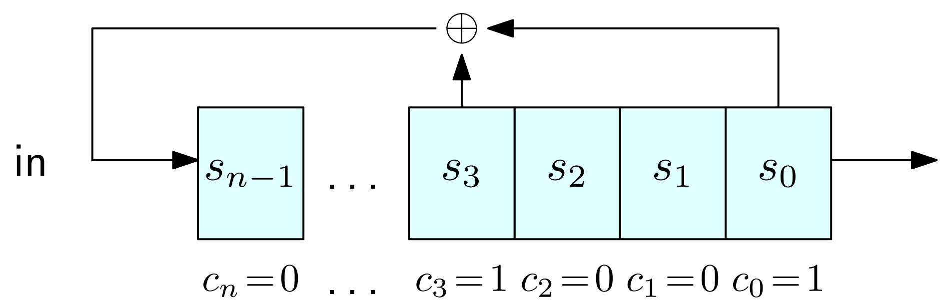
The subset of bits that are XOR-ed together can be described by n coefficients $c_0, c_1, \ldots, c_{n-1}$



- If $c_i = 0$ then s_i is ignored (equivalently, 0 is XOR-ed in place of s_i)

Linear Feedback Shift Registers (LFSR)

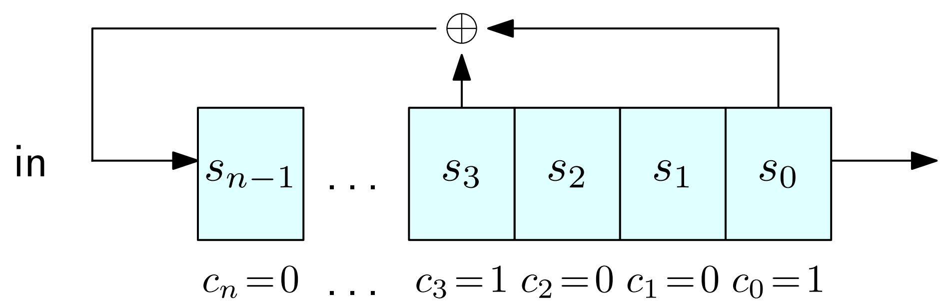
The subset of bits that are XOR-ed together can be described by n coefficients $c_0, c_1, \ldots, c_{n-1}$



- If $c_i = 0$ then s_i is ignored (equivalently, 0 is XOR-ed in place of s_i)
- If $c_i = 1$ then s_i is XOR-ed

Linear Feedback Shift Registers (LFSR)

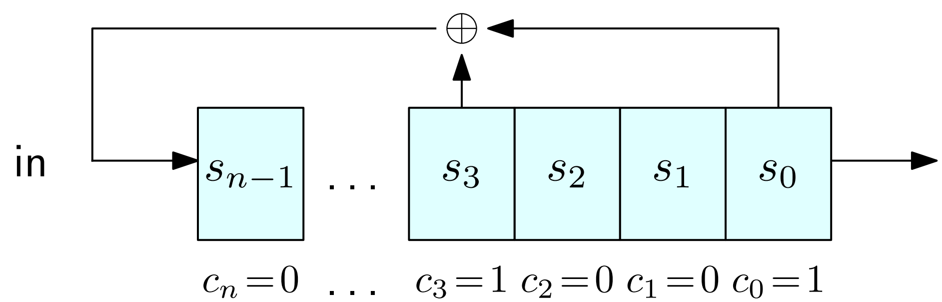
The subset of bits that are XOR-ed together can be described by n coefficients $c_0, c_1, \ldots, c_{n-1}$



- If $c_i = 0$ then s_i is ignored (equivalently, 0 is XOR-ed in place of s_i) $\quad 0 = c_i s_i$
- If $c_i = 1$ then s_i is XOR-ed $\quad s_i = c_i s_i$

Linear Feedback Shift Registers (LFSR)

The subset of bits that are XOR-ed together can be described by n coefficients $c_0, c_1, \ldots, c_{n-1}$

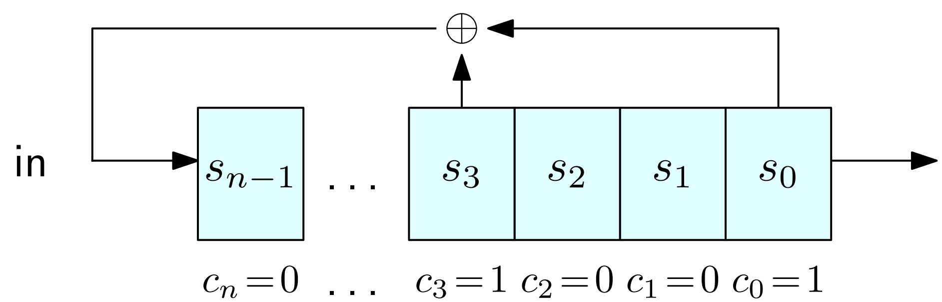


- If $c_i = 0$ then s_i is ignored (equivalently, 0 is XOR-ed in place of s_i) $\quad 0 = c_i s_i$
- If $c_i = 1$ then s_i is XOR-ed $\quad s_i = c_i s_i$

At each clock tick, the state is updated from $s_{n-1} \ldots s_1 s_0$ to $s'_{n-1} \ldots s'_1 s'_0$:

Linear Feedback Shift Registers (LFSR)

The subset of bits that are XOR-ed together can be described by n coefficients $c_0, c_1, \ldots, c_{n-1}$



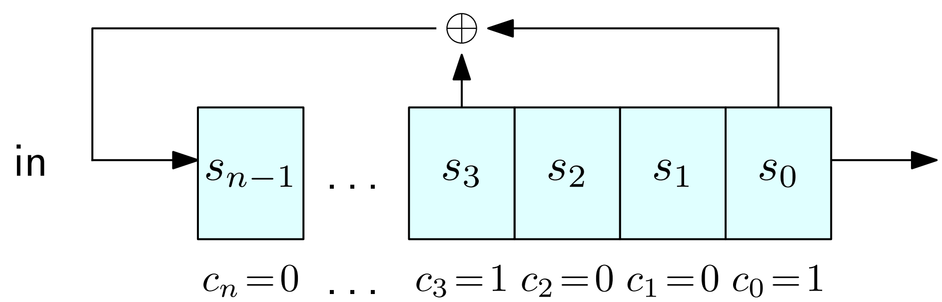
- If $c_i = 0$ then s_i is ignored (equivalently, 0 is XOR-ed in place of s_i) $0 = c_i s_i$
- If $c_i = 1$ then s_i is XOR-ed $s_i = c_i s_i$

At each clock tick, the state is updated from $s_{n-1} \ldots s_1 s_0$ to $s'_{n-1} \ldots s'_1 s'_0$:

- $s'_i = s_{i+1}$ for $i < n-1$
- $s'_{n-1} = \bigoplus_{i=0}^{n-1} c_i s_i$

Linear Feedback Shift Registers (LFSR)

The subset of bits that are XOR-ed together can be described by n coefficients $c_0, c_1, \ldots, c_{n-1}$



- If $c_i = 0$ then s_i is ignored (equivalently, 0 is XOR-ed in place of s_i) $\quad 0 = c_i s_i$
- If $c_i = 1$ then s_i is XOR-ed $\quad s_i = c_i s_i$

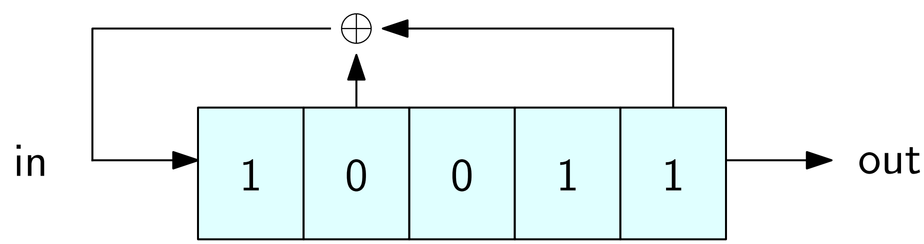
At each clock tick, the state is updated from $s_{n-1} \ldots s_1 s_0$ to $s'_{n-1} \ldots s'_1 s'_0$:

- $s'_i = s_{i+1}$ for $i < n - 1$
- $s'_{n-1} = \bigoplus_{i=0}^{n-1} c_i s_i$

The coefficients are part of the construction of the LFSR.

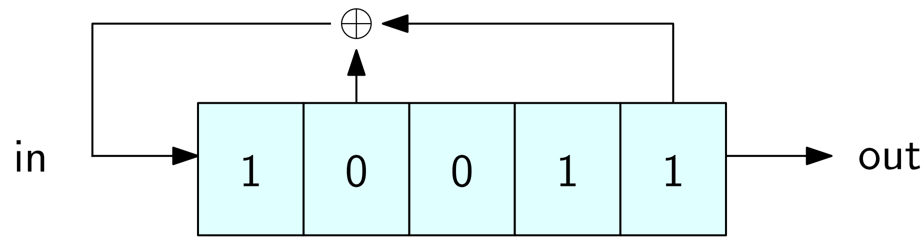
By Kerckhoffs' principle they should not be considered secret

LFSRs as stream ciphers



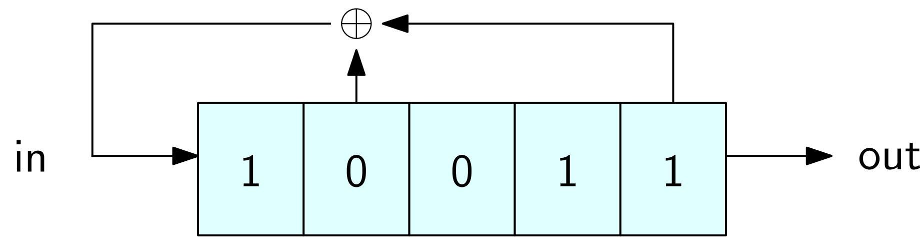
- Init(s): set the bits of the shift register to the bits of s

LFSRs as stream ciphers



- Init(s): set the bits of the shift register to the bits of s
- Next(st): corresponds to one clock tick from state st. It returns the bit from the "out" line, and the new state after the shift

LFSRs as stream ciphers

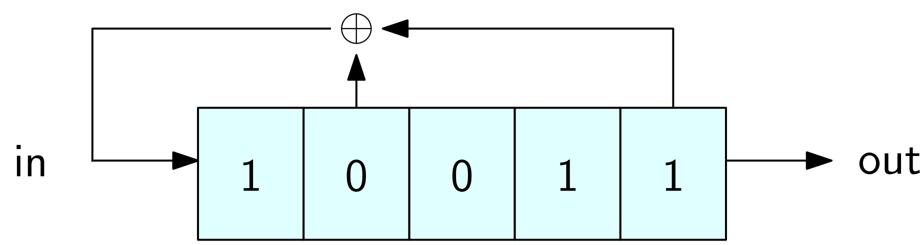


- Init(s): set the bits of the shift register to the bits of s

- Next(st): corresponds to one clock tick from state st. It returns the bit from the "out" line, and the new state after the shift

Since the number of states is finite (i.e., 2^n), the output sequence must eventually repeat

LFSRs as stream ciphers



- Init(s): set the bits of the shift register to the bits of s

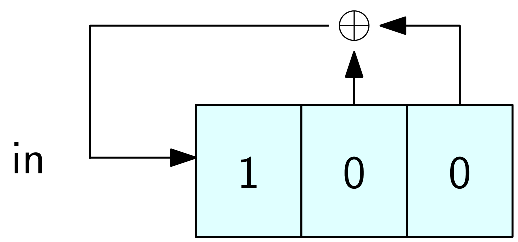
- Next(st): corresponds to one clock tick from state st. It returns the bit from the "out" line, and the new state after the shift

Since the number of states is finite (i.e., 2^n), the output sequence must eventually repeat

A necessary (but not sufficient) condition for stream ciphers to be secure is that the time it takes for repeats to happen must be long

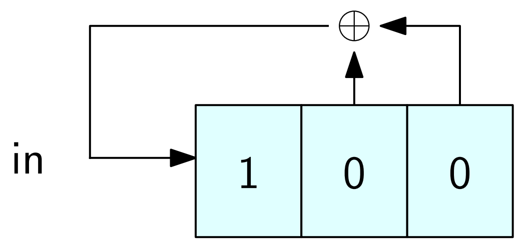
State Graph

Given a FSR



State Graph

Given a FSR

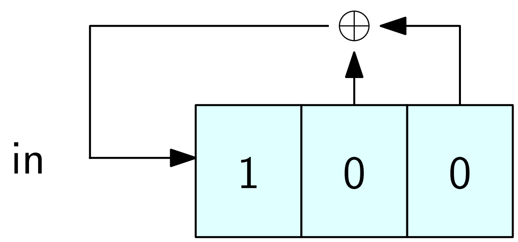


We can create a **state graph** $G = (V, E)$ in which each vertex is a state, i.e., $V = \{0, 1\}^n \ldots$

$\ldots$ and there is a directed edge labelled $y \in \{0, 1\}$ from st to st$'$ iff Next(sf) $= (y, \mathsf{sf}')$.

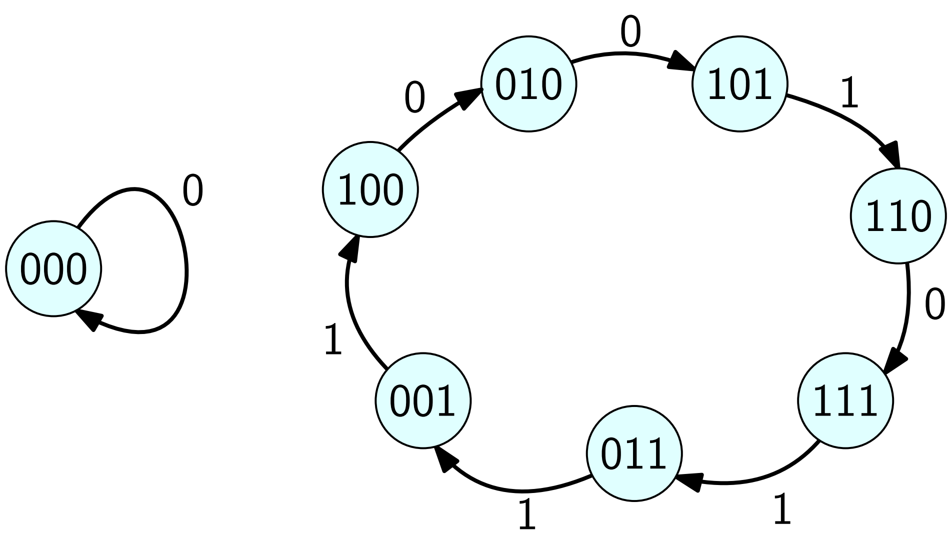
State Graph

Given a FSR



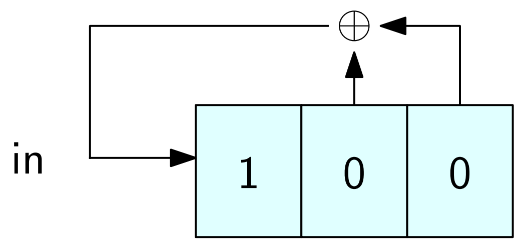
We can create a **state graph** $G = (V, E)$ in which each vertex is a state, i.e., $V = \{0,1\}^n \ldots$

$\ldots$ and there is a directed edge labelled $y \in \{0,1\}$ from st to st$'$ iff Next(sf) $= (y, \mathsf{sf}')$.



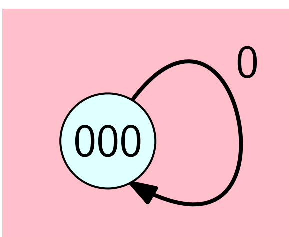
State Graph

Given a FSR

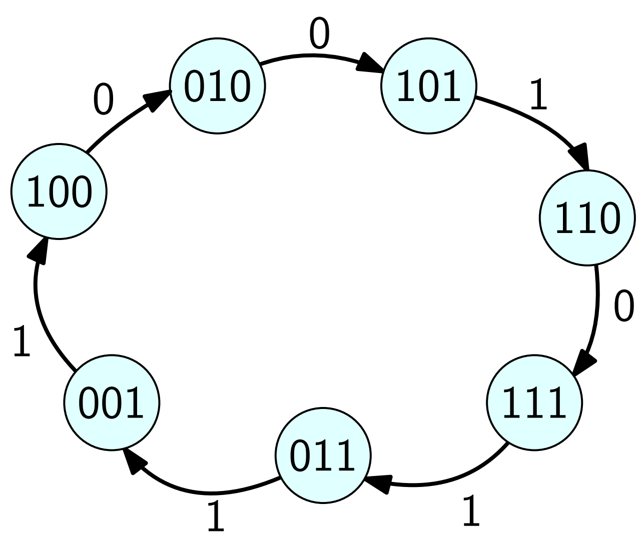


We can create a **state graph** $G = (V, E)$ in which each vertex is a state, i.e., $V = \{0,1\}^n \ldots$

$\ldots$ and there is a directed edge labelled $y \in \{0,1\}$ from st to st′ iff $\text{Next}(\text{sf}) = (y, \text{sf}')$.

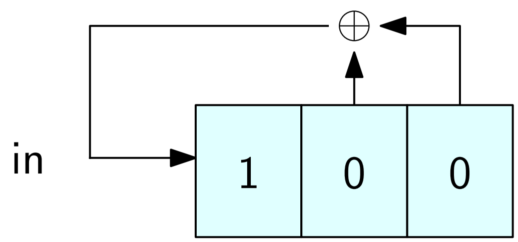


In a LFSR, state $00\ldots0$ always has a self-loop



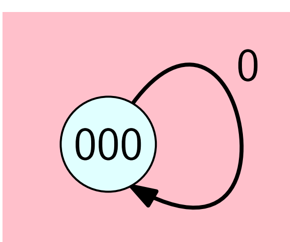
State Graph

Given a FSR

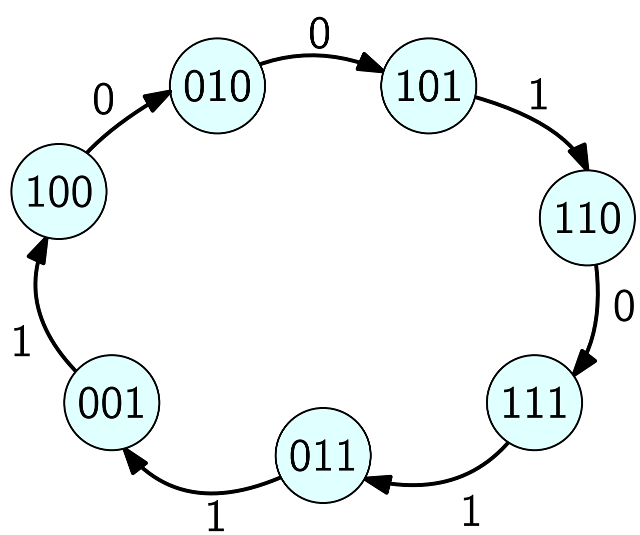


We can create a **state graph** $G = (V, E)$ in which each vertex is a state, i.e., $V = \{0,1\}^n \ldots$

$\ldots$ and there is a directed edge labelled $y \in \{0,1\}$ from st to st$'$ iff Next(sf) $= (y, \text{sf}')$.



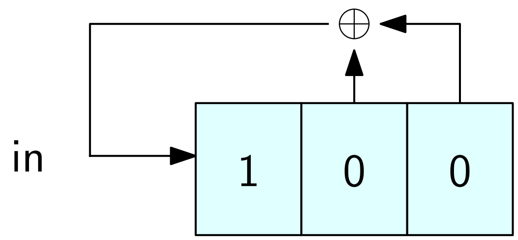
In a LFSR, state $00\ldots0$ always has a self-loop



A LFSR with degree n is a **maximum length** LFSR if its state graph has a cycle though all $2^n - 1$ non-zero states.

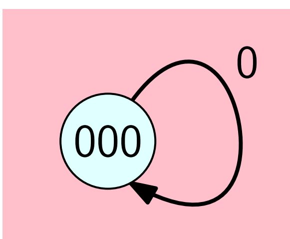
State Graph

Given a FSR

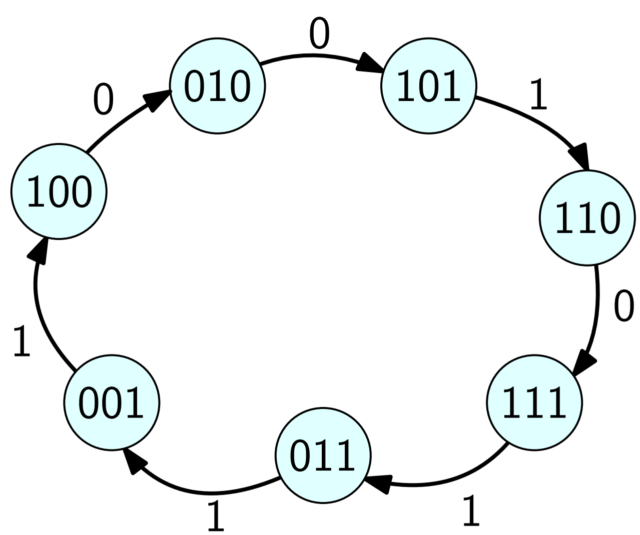


We can create a **state graph** $G = (V, E)$ in which each vertex is a state, i.e., $V = \{0,1\}^n \ldots$

$\ldots$ and there is a directed edge labelled $y \in \{0,1\}$ from st to st′ iff $\mathsf{Next}(\mathsf{sf}) = (y, \mathsf{sf}')$.



In a LFSR, state $00\ldots0$ always has a self-loop



A LFSR with degree n is a **maximum length** LFSR if its state graph has a cycle though all $2^n - 1$ non-zero states.

For any n, it is known how to set the coefficients to obtain a maximum length LFSR of degree n

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \ldots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \ldots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \ldots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

One might try to use the seed to set the feedback coefficients...

- The adversary can still recover all the coefficients!

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \ldots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

One might try to use the seed to set the feedback coefficients...

- The adversary can still recover all the coefficients!
- The adversary also observes the second group of n bits $y_n y_{n+1} y_{n+2} \ldots y_{2n-1}$ output by the LFSR

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \ldots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

One might try to use the seed to set the feedback coefficients...

- The adversary can still recover all the coefficients!
- The adversary also observes the second group of n bits $y_n y_{n+1} y_{n+2} \ldots y_{2n-1}$ output by the LFSR

$$y_n = c_{n-1} y_{n-1} \oplus c_{n-2} y_{n-2} \oplus \cdots \oplus c_0 y_0$$

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \ldots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

One might try to use the seed to set the feedback coefficients...

- The adversary can still recover all the coefficients!
- The adversary also observes the second group of n bits $y_n y_{n+1} y_{n+2} \ldots y_{2n-1}$ output by the LFSR

$$\begin{cases} y_n = c_{n-1} y_{n-1} \oplus c_{n-2} y_{n-2} \oplus \cdots \oplus c_0 y_0 \\ y_{n+1} = c_{n-1} y_n \oplus c_{n-2} y_{n-1} \oplus \cdots \oplus c_0 y_1 \\ \\ \\ \end{cases}$$

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \ldots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

One might try to use the seed to set the feedback coefficients...

- The adversary can still recover all the coefficients!
- The adversary also observes the second group of n bits $y_n y_{n+1} y_{n+2} \ldots y_{2n-1}$ output by the LFSR

$$\begin{cases} y_n = c_{n-1} y_{n-1} \oplus c_{n-2} y_{n-2} \oplus \cdots \oplus c_0 y_0 \\ y_{n+1} = c_{n-1} y_n \oplus c_{n-2} y_{n-1} \oplus \cdots \oplus c_0 y_1 \\ y_{n+2} = c_{n-1} y_{n+1} \oplus c_{n-2} y_n \oplus \cdots \oplus c_0 y_2 \\ \vdots \\ y_{2n-1} = c_{n-1} y_{2n-2} \oplus c_{n-2} y_{2n-3} \oplus \cdots \oplus c_0 y_{n-1} \end{cases}$$

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \dots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

One might try to use the seed to set the feedback coefficients...

- The adversary can still recover all the coefficients!
- The adversary also observes the second group of n bits $y_n y_{n+1} y_{n+2} \dots y_{2n-1}$ output by the LFSR

$$\begin{cases} y_n = c_{n-1} y_{n-1} \oplus c_{n-2} y_{n-2} \oplus \cdots \oplus c_0 y_0 \\ y_{n+1} = c_{n-1} y_n \oplus c_{n-2} y_{n-1} \oplus \cdots \oplus c_0 y_1 \\ y_{n+2} = c_{n-1} y_{n+1} \oplus c_{n-2} y_n \oplus \cdots \oplus c_0 y_2 \\ \qquad \vdots \\ y_{2n-1} = c_{n-1} y_{2n-2} \oplus c_{n-2} y_{2n-3} \oplus \cdots \oplus c_0 y_{n-1} \end{cases}$$

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \dots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

One might try to use the seed to set the feedback coefficients...

- The adversary can still recover all the coefficients!
- The adversary also observes the second group of n bits $y_n y_{n+1} y_{n+2} \dots y_{2n-1}$ output by the LFSR

$$\begin{cases} y_n &= c_{n-1} y_{n-1} \oplus c_{n-2} y_{n-2} \oplus \cdots \oplus c_0 y_0 \\ y_{n+1} &= c_{n-1} y_n \oplus c_{n-2} y_{n-1} \oplus \cdots \oplus c_0 y_1 \\ y_{n+2} &= c_{n-1} y_{n+1} \oplus c_{n-2} y_n \oplus \cdots \oplus c_0 y_2 \\ & \vdots \\ y_{2n-1} &= c_{n-1} y_{2n-2} \oplus c_{n-2} y_{2n-3} \oplus \cdots \oplus c_0 y_{n-1} \end{cases}$$

- n linear equations (over $\mathbb{Z}_2$)
- n variables
- If the LFSR has maximum length*, then the equations are linearly independent

*If the LFSR does not have maximum length, variants of the attack still apply

Key recovery attacks on LFSRs

Linear Feedback Shift Registers are **not** secure stream ciphers!

- The first n bits $y_0 y_1 y_2 \ldots y_{n-1}$ output by the LFSR are exactly the seed (right to left)!
- Since the coefficients are known, an adversary can generate the same stream of bits.

One might try to use the seed to set the feedback coefficients...

- The adversary can still recover all the coefficients!
- The adversary also observes the second group of n bits $y_n y_{n+1} y_{n+2} \ldots y_{2n-1}$ output by the LFSR

$$
\begin{cases}
y_n = c_{n-1} y_{n-1} \oplus c_{n-2} y_{n-2} \oplus \cdots \oplus c_0 y_0 \\
y_{n+1} = c_{n-1} y_n \oplus c_{n-2} y_{n-1} \oplus \cdots \oplus c_0 y_1 \\
y_{n+2} = c_{n-1} y_{n+1} \oplus c_{n-2} y_n \oplus \cdots \oplus c_0 y_2 \\
\vdots \\
y_{2n-1} = c_{n-1} y_{2n-2} \oplus c_{n-2} y_{2n-3} \oplus \cdots \oplus c_0 y_{n-1}
\end{cases}
$$

- n linear equations (over $\mathbb{Z}_2$)
- n variables
- If the LFSR has maximum length*, then the equations are linearly independent

$\Downarrow$

Unique solution! Solve the system and recover all coefficients

*If the LFSR does not have maximum length, variants of the attack still apply

Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:

$$\begin{array}{c} y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 \\ 0, 0, 1, 1, 1, 1, 0, 1 \end{array}$$

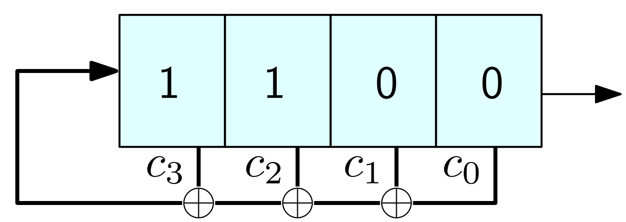
Recover the length and the coefficients of the LFSR

Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:

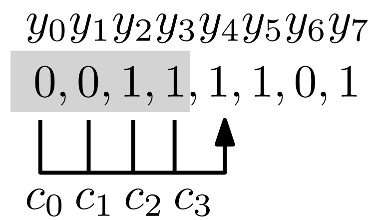
$$\begin{array}{cccccccc} y_0 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\ 0, & 0, & 1, & 1, & 1, & 1, & 0, & 1 \end{array}$$

Recover the length and the coefficients of the LFSR

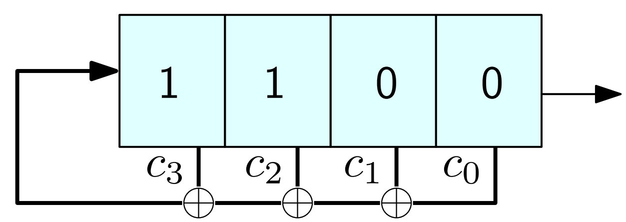


Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:

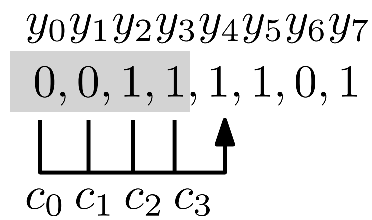


Recover the length and the coefficients of the LFSR



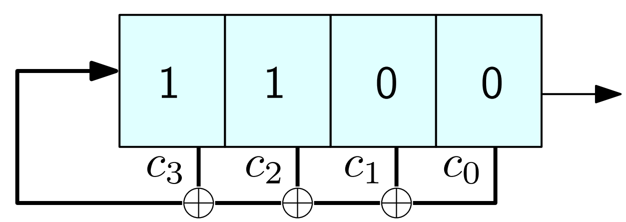
Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:



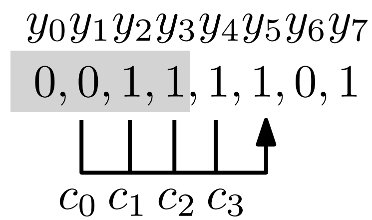
Recover the length and the coefficients of the LFSR

$$\begin{cases} 1 = c_2 \oplus c_3 \\ \\ \\ \end{cases}$$



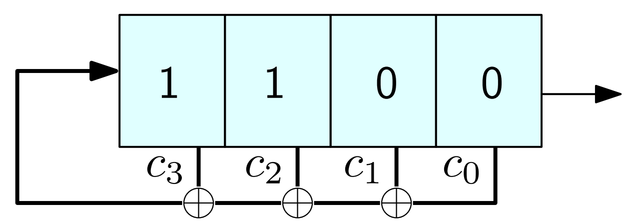
Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:



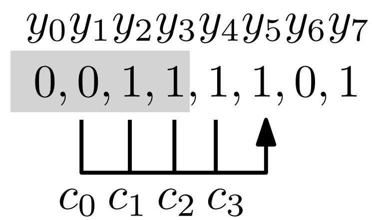
Recover the length and the coefficients of the LFSR

$$\begin{cases} 1 = c_2 \oplus c_3 \\ \\ \\ \end{cases}$$



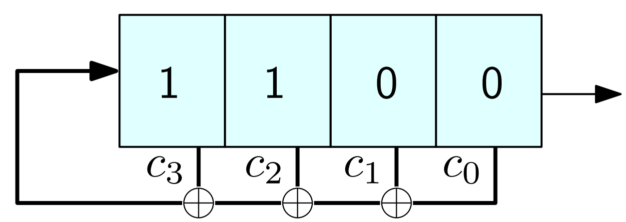
Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:



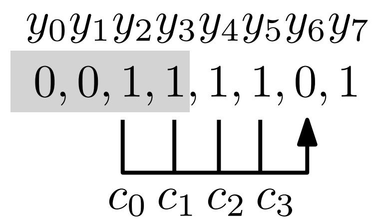
Recover the length and the coefficients of the LFSR

$$
\begin{cases}
1 = c_2 \oplus c_3 \\
1 = c_1 \oplus c_2 \oplus c_3 \\
\\
\end{cases}
$$



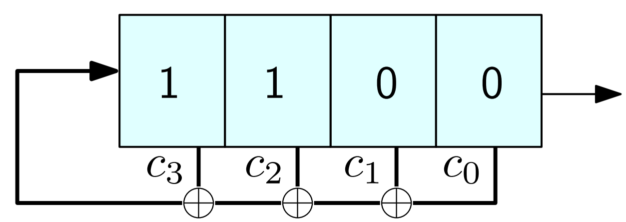
Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:



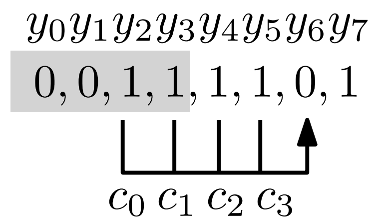
Recover the length and the coefficients of the LFSR

$$\begin{cases} 1 = c_2 \oplus c_3 \\ 1 = c_1 \oplus c_2 \oplus c_3 \\ \\ \\ \end{cases}$$



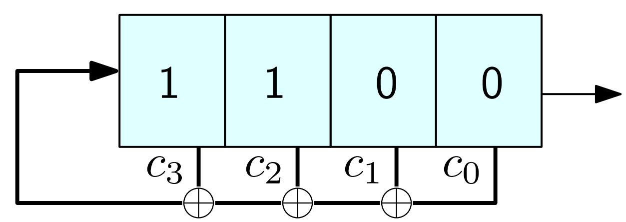
Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:



Recover the length and the coefficients of the LFSR

$$\begin{cases} 1 = c_2 \oplus c_3 \\ 1 = c_1 \oplus c_2 \oplus c_3 \\ 0 = c_0 \oplus c_1 \oplus c_2 \oplus c_3 \\ \end{cases}$$



Key recovery attacks on LFSRs: example

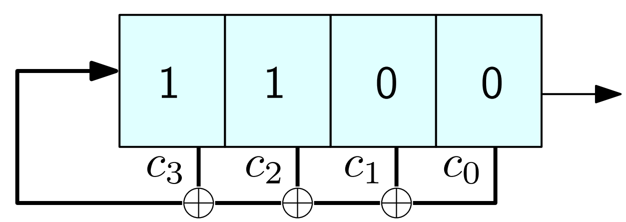
The output of a maximum-length LFSR of degree 4 is:

$$\begin{array}{cccccccc} y_0 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\ 0, & 0, & 1, & 1, & 1, & 1, & 0, & 1 \end{array}$$

$$c_0 \; c_1 \; c_2 \; c_3$$

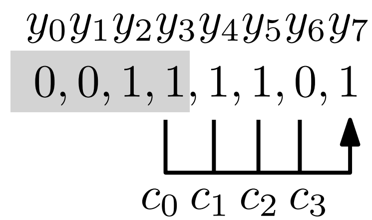
Recover the length and the coefficients of the LFSR

$$\begin{cases} 1 = c_2 \oplus c_3 \\ 1 = c_1 \oplus c_2 \oplus c_3 \\ 0 = c_0 \oplus c_1 \oplus c_2 \oplus c_3 \\ \end{cases}$$



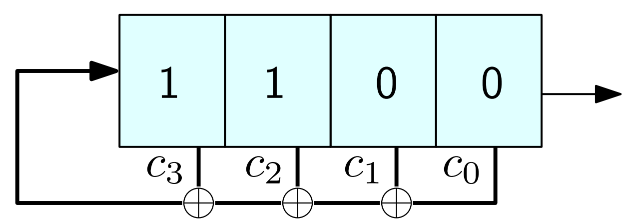
Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:



Recover the length and the coefficients of the LFSR

$$
\begin{cases}
1 = c_2 \oplus c_3 \\
1 = c_1 \oplus c_2 \oplus c_3 \\
0 = c_0 \oplus c_1 \oplus c_2 \oplus c_3 \\
1 = c_0 \oplus c_1 \oplus c_2
\end{cases}
$$



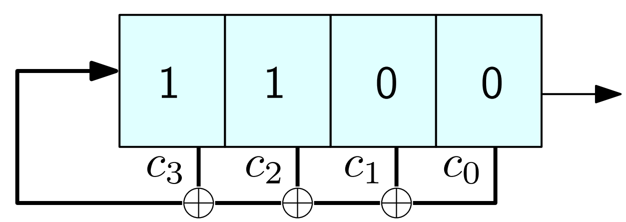
Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:

$$\begin{array}{cccccccc} y_0 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\ 0, & 0, & 1, & 1, & 1, & 1, & 0, & 1 \end{array}$$

Recover the length and the coefficients of the LFSR

$$\begin{cases} 1 = c_2 \oplus c_3 \\ 1 = c_1 \oplus c_2 \oplus c_3 \\ 0 = c_0 \oplus c_1 \oplus c_2 \oplus c_3 \\ 1 = c_0 \oplus c_1 \oplus c_2 \end{cases} \implies \begin{cases} c_0 = 1 \\ c_1 = 0 \\ c_2 = 0 \\ c_3 = 1 \end{cases}$$



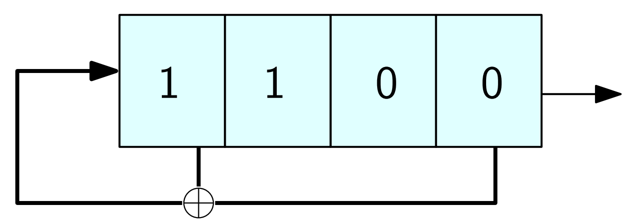
Key recovery attacks on LFSRs: example

The output of a maximum-length LFSR of degree 4 is:

$$\begin{array}{cccccccc} y_0 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\ 0, & 0, & 1, & 1, & 1, & 1, & 0, & 1 \end{array}$$

Recover the length and the coefficients of the LFSR

$$\begin{cases} 1 = c_2 \oplus c_3 \\ 1 = c_1 \oplus c_2 \oplus c_3 \\ 0 = c_0 \oplus c_1 \oplus c_2 \oplus c_3 \\ 1 = c_0 \oplus c_1 \oplus c_2 \end{cases} \implies \begin{cases} c_0 = 1 \\ c_1 = 0 \\ c_2 = 0 \\ c_3 = 1 \end{cases}$$



Nonlinear Feedback Shift Registers (NLFSRs)

To avoid this weakness, we need to add some non-linear component

Nonlinear Feedback Shift Registers (NLFSRs)

To avoid this weakness, we need to add some non-linear component

Multiple options:

- **Nonlinear feedback:** on state update, choose the new value of the leftmost register s_{n-1} as some non-linear function $g(s_0, s_1, \ldots, s_{n-1})$ of the current registers

Nonlinear Feedback Shift Registers (NLFSRs)

To avoid this weakness, we need to add some non-linear component

Multiple options:

- **Nonlinear feedback:** on state update, choose the new value of the leftmost register s_{n-1} as some non-linear function $g(s_0, s_1, \ldots, s_{n-1})$ of the current registers

- **Nonlinear output:** the output bit is some function $g(s_0, s_1, \ldots, s_{n-1})$ of the current state (rather than simply s_0)

Nonlinear Feedback Shift Registers (NLFSRs)

To avoid this weakness, we need to add some non-linear component

Multiple options:

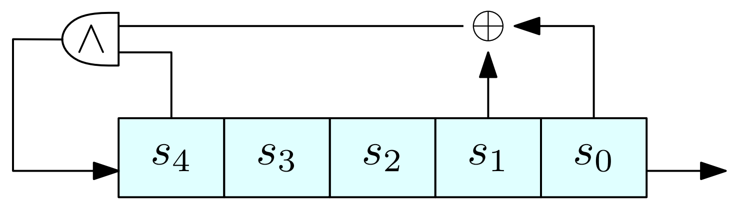
- **Nonlinear feedback:** on state update, choose the new value of the leftmost register s_{n-1} as some non-linear function $g(s_0, s_1, \ldots, s_{n-1})$ of the current registers

- **Nonlinear output:** the output bit is some function $g(s_0, s_1, \ldots, s_{n-1})$ of the current state (rather than simply s_0)

- **Combination generators:** use multiple LFSRs and combine their outputs in some nonlinear way

Adding nonlinearity

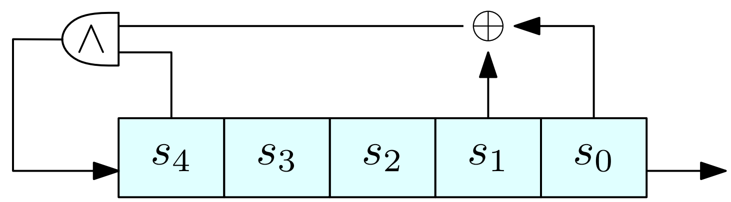
Nonlinear feedback: on state update, choose the new value of the leftmost register s_{n-1} as some non-linear function $g(s_0, s_1, \ldots, s_{n-1})$ of the current registers



- In the example: $g(s_0, s_1, \ldots, s_{n-1}) = (s_0 \oplus s_1) \wedge s_4$

Adding nonlinearity

Nonlinear feedback: on state update, choose the new value of the leftmost register s_{n-1} as some non-linear function $g(s_0, s_1, \ldots, s_{n-1})$ of the current registers

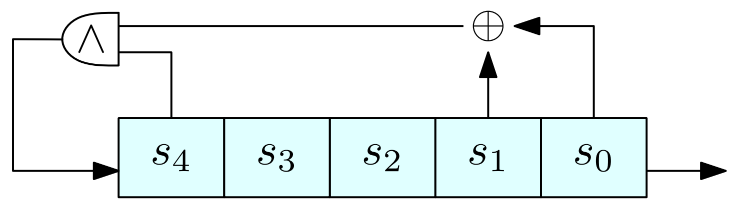


- In the example: $g(s_0, s_1, \ldots, s_{n-1}) = (s_0 \oplus s_1) \wedge s_4$

- Care must be taken to ensure that $g(\cdot) = 1$ with probability $\approx \frac{1}{2}$

Adding nonlinearity

Nonlinear feedback: on state update, choose the new value of the leftmost register s_{n-1} as some non-linear function $g(s_0, s_1, \ldots, s_{n-1})$ of the current registers



- In the example: $g(s_0, s_1, \ldots, s_{n-1}) = (s_0 \oplus s_1) \wedge s_4$

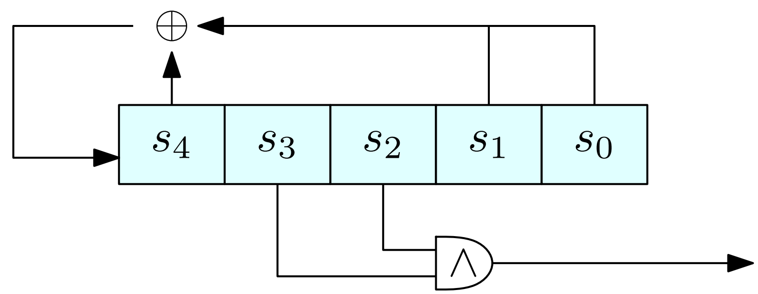
- Care must be taken to ensure that $g(\cdot) = 1$ with probability $\approx \frac{1}{2}$

  The function g above is not a great choice, since its is 0 whenever at least one of $s_0 \oplus s_1$ and s_4 is 0

  If we heuristically think of the state as a uniformly random string, then $g(\cdot)$ will be zero 75% of the time!

Adding nonlinearity

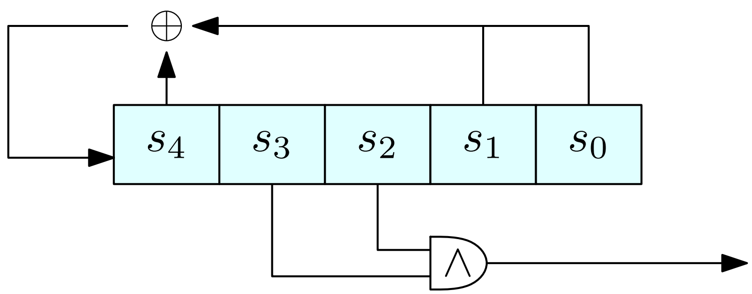
Nonlinear output: the output bit is some function $g(s_0, s_1, \ldots, s_{n-1})$ of the current state (rather than simply s_0)



In the example: $g(s_0, s_1, \ldots, s_{n-1}) = s_2 \wedge s_3$

Adding nonlinearity

Nonlinear output: the output bit is some function $g(s_0, s_1, \ldots, s_{n-1})$ of the current state (rather than simply s_0)

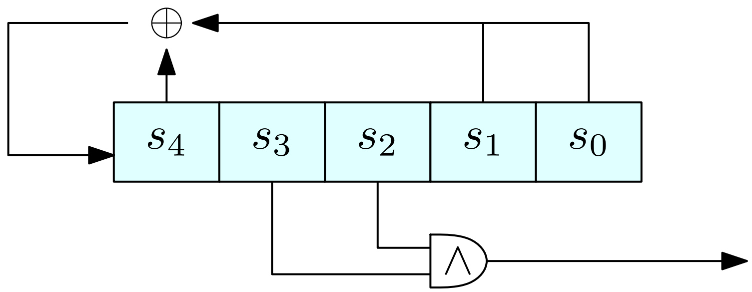


In the example: $g(s_0, s_1, \ldots, s_{n-1}) = s_2 \wedge s_3$

- The function $g(\cdot)$ is called **filter**

Adding nonlinearity

Nonlinear output: the output bit is some function $g(s_0, s_1, \ldots, s_{n-1})$ of the current state (rather than simply s_0)

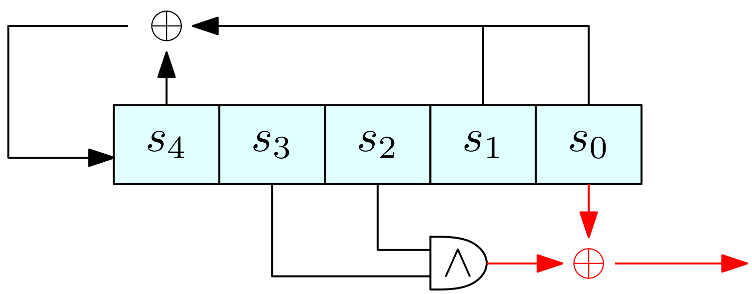


In the example: $g(s_0, s_1, \ldots, s_{n-1}) = s_2 \wedge s_3$

- The function $g(\cdot)$ is called **filter**
- Care must be taken to ensure that $g(\cdot) = 1$ with probability $\approx \frac{1}{2}$

Adding nonlinearity

Nonlinear output: the output bit is some function $g(s_0, s_1, \ldots, s_{n-1})$ of the current state (rather than simply s_0)

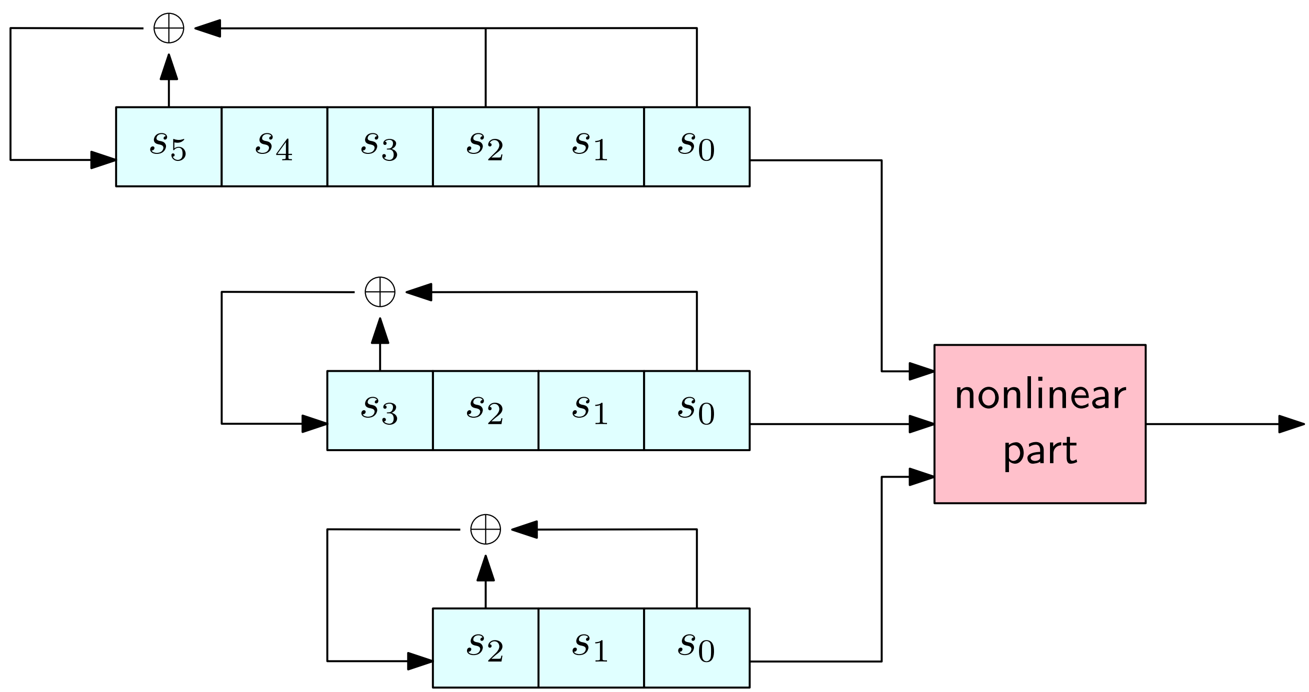


In the example: $g(s_0, s_1, \ldots, s_{n-1}) = s_2 \wedge s_3$

- The function $g(\cdot)$ is called **filter**
- Care must be taken to ensure that $g(\cdot) = 1$ with probability $\approx \frac{1}{2}$
- A better function: $g(s_0, s_1, \ldots, s_{n-1}) = (s_2 \wedge s_3) \oplus s_0$

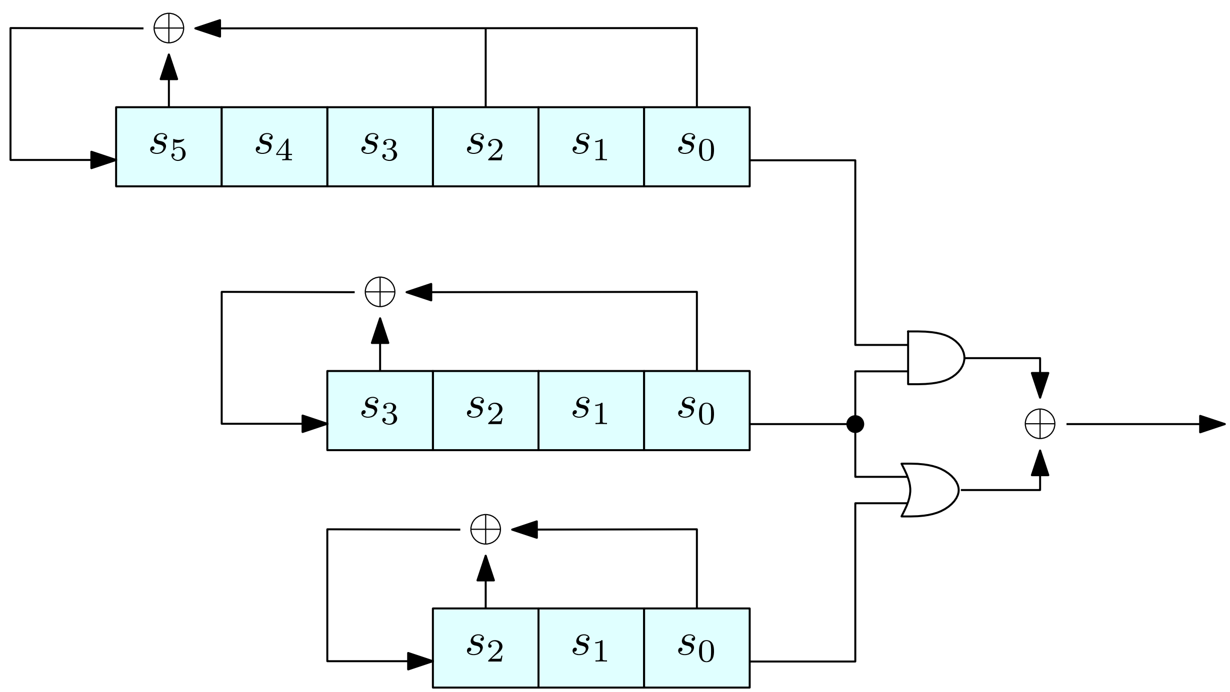
Adding nonlinearity

Combination generators: use multiple LFSRs and combine their outputs in some nonlinear way



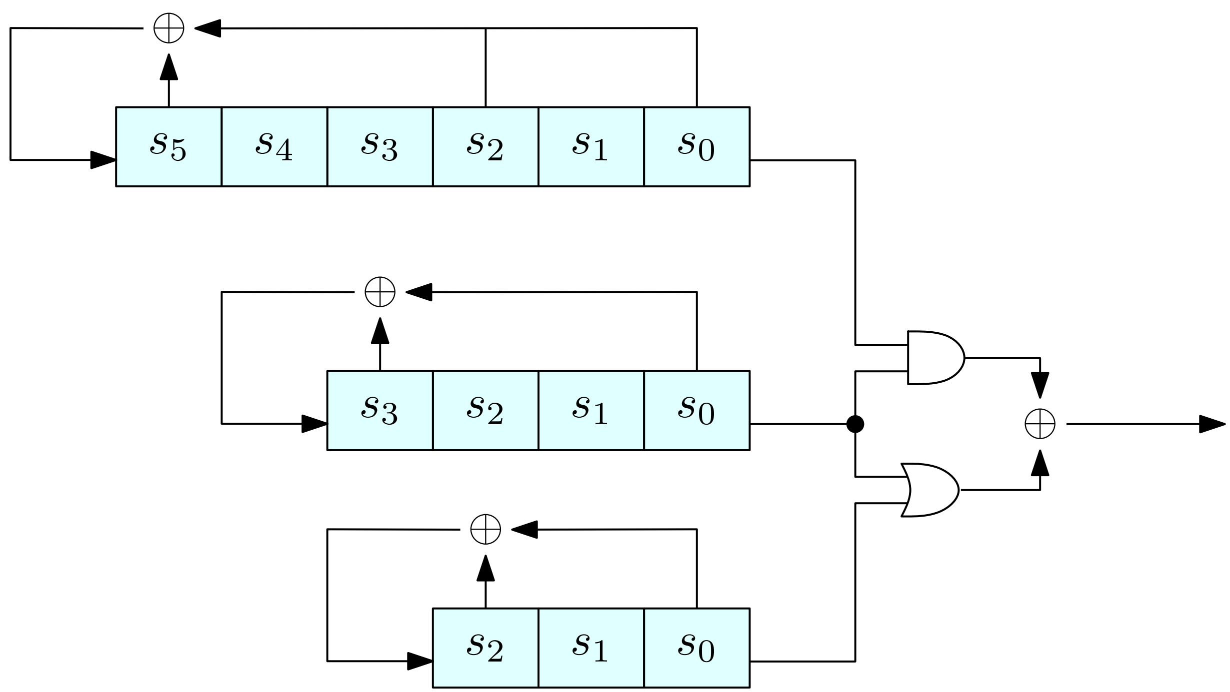
Adding nonlinearity

Combination generators: use multiple LFSRs and combine their outputs in some nonlinear way



Adding nonlinearity

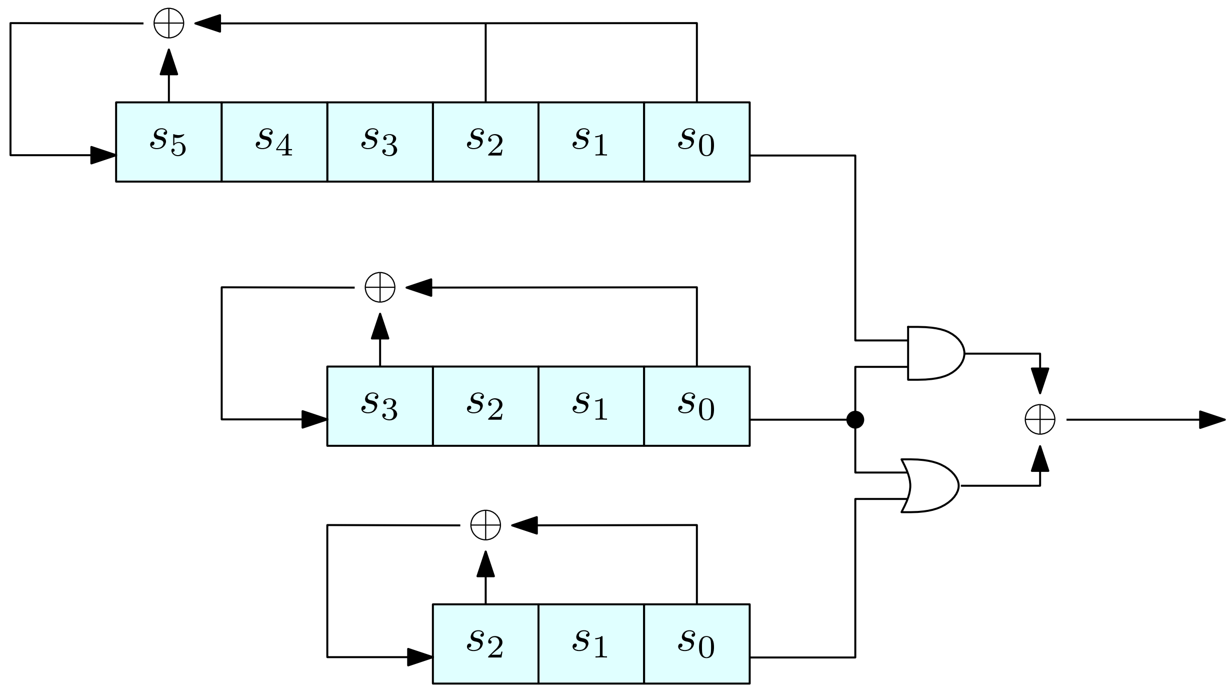
Combination generators: use multiple LFSRs and combine their outputs in some nonlinear way



- The LFSRs do not need to have the same degrees (in fact, it is better if they have different degrees)

Adding nonlinearity

Combination generators: use multiple LFSRs and combine their outputs in some nonlinear way

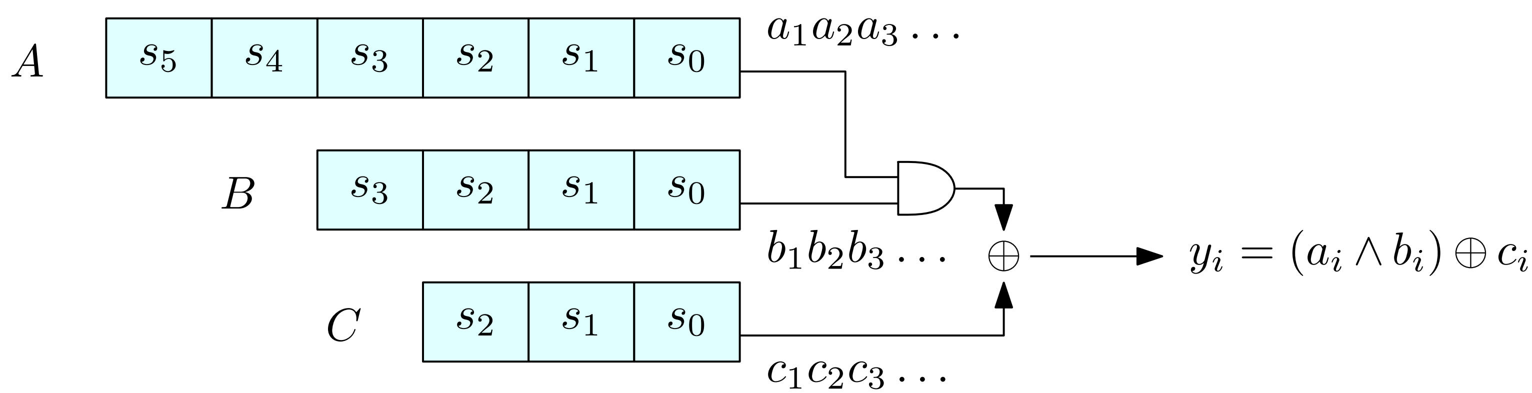


- The LFSRs do not need to have the same degrees (in fact, it is better if they have different degrees)
- Ideally, if the degrees are $d_1, d_2, d_3, \ldots$, we would like attacks to take time $\approx 2^{d_1+d_2+d_3+\cdots}$

Correlation attacks on combination generators

Care must be taken to ensure that the output bit is not biased towards the output of any of the LFSRs

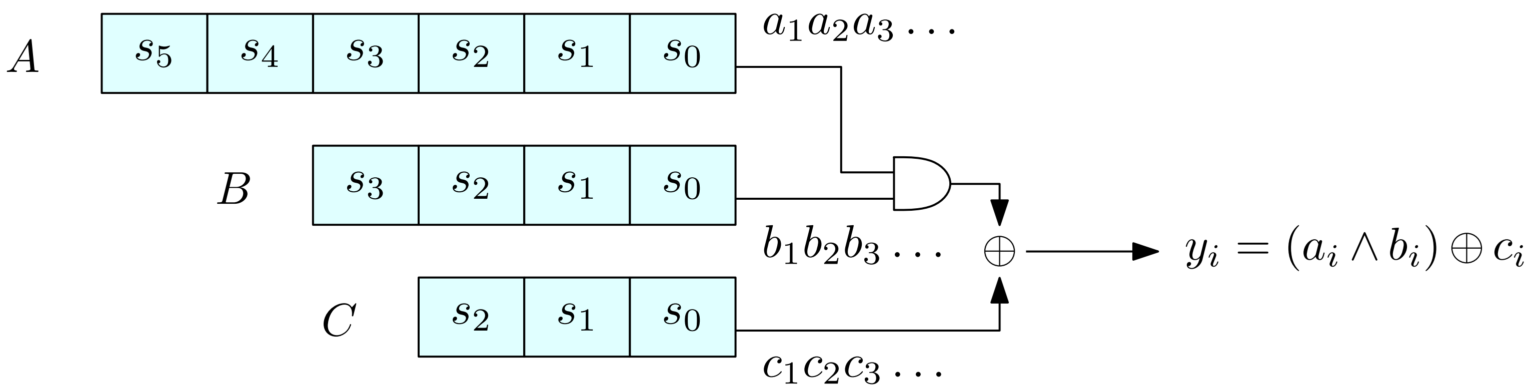
A bad example:



Correlation attacks on combination generators

Care must be taken to ensure that the output bit is not biased towards the output of any of the LFSRs

A bad example:

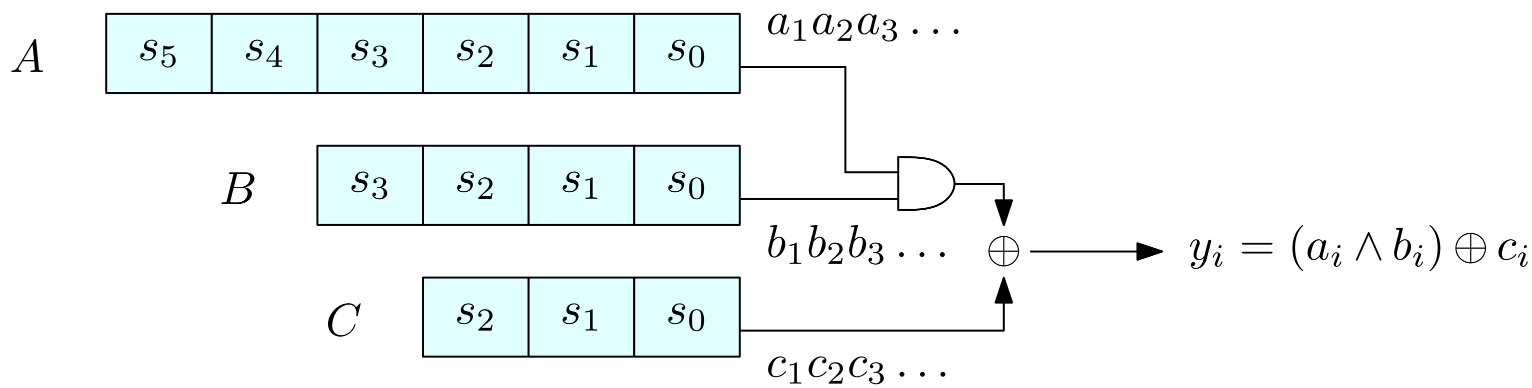


- 75% of the time $(a_i \wedge b_i)$ is 0
- When this happens, $y_i = c_i$

Correlation attacks on combination generators

Care must be taken to ensure that the output bit is not biased towards the output of any of the LFSRs

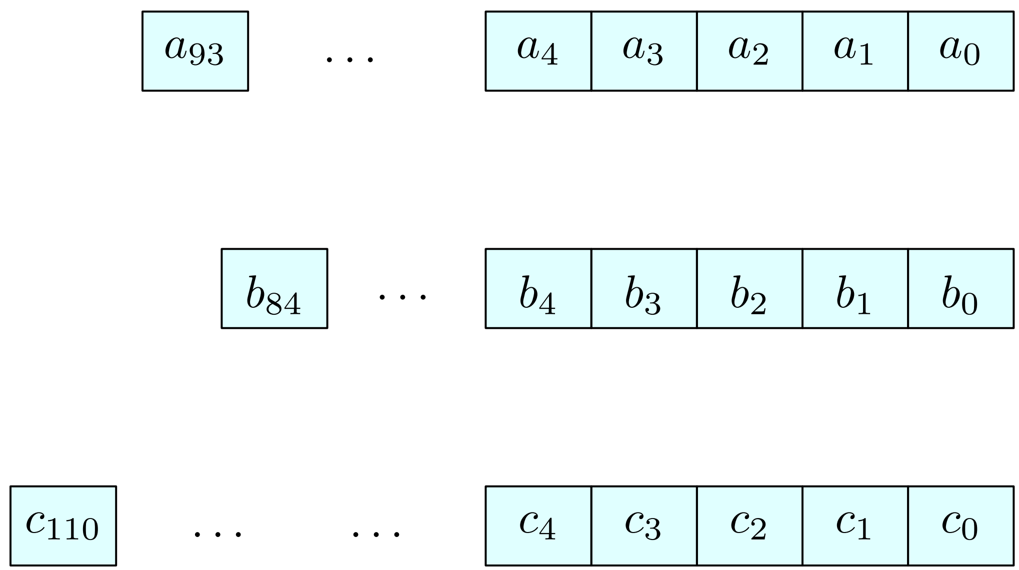
A bad example:



- 75% of the time $(a_i \wedge b_i)$ is 0
- When this happens, $y_i = c_i$
- We can run a bruteforce attack on C:
  - Try all possible initial states. For every state generate a stream of bits $c'_1, c'_2, c'_3, \dots$
  - When the initial state is correct, $\approx 3/4$ of the bits c_is match with the corresponding c'_is

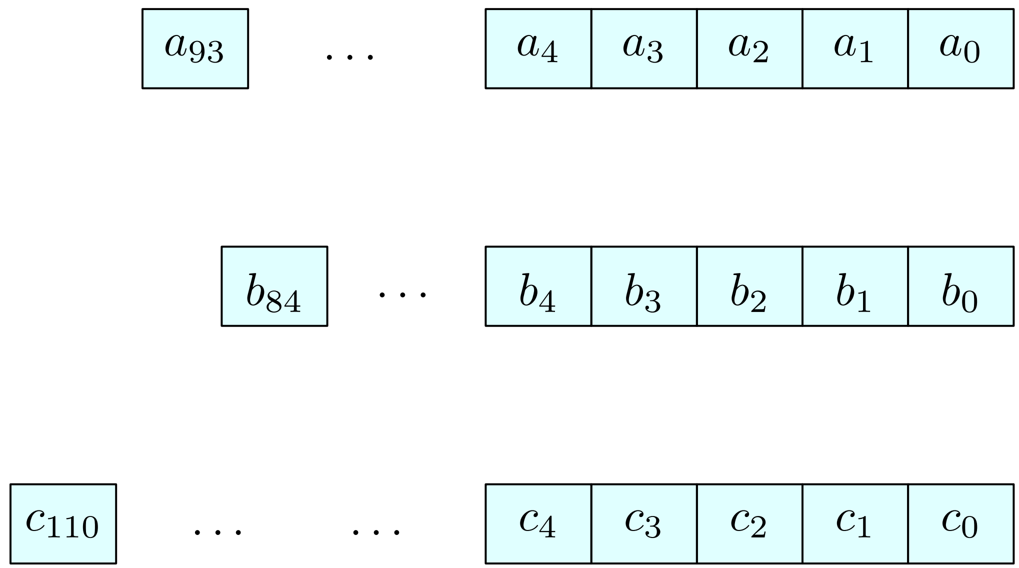
Back to Trivium

- Three FSRs (say A, B, C) of degrees 93, 84, and 111 (overall, the state is 288 bits long)



Back to Trivium

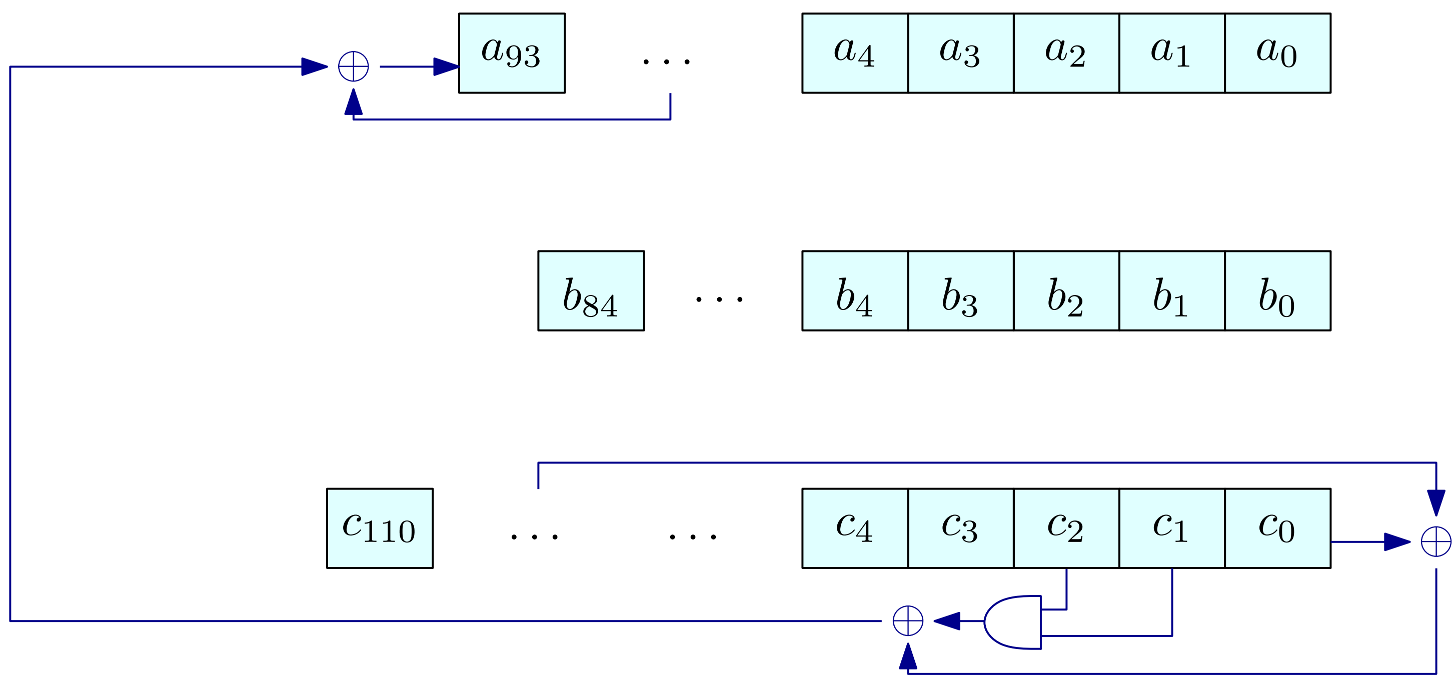
- Three FSRs (say A, B, C) of degrees 93, 84, and 111 (overall, the state is 288 bits long)



- The FSRs are **coupled**: the input of each FSR is a non-linear function of a register from that FSR, and of 4 registers from another FSR

Back to Trivium

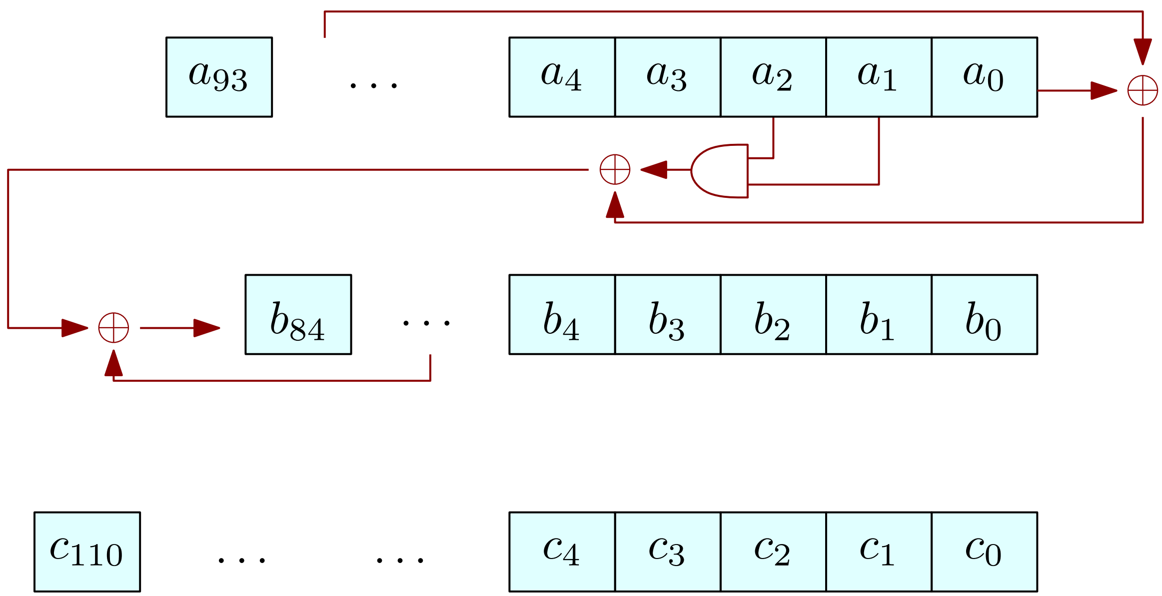
- Three FSRs (say A, B, C) of degrees 93, 84, and 111 (overall, the state is 288 bits long)



- The FSRs are **coupled**: the input of each FSR is a non-linear function of a register from that FSR, and of 4 registers from another FSR

Back to Trivium

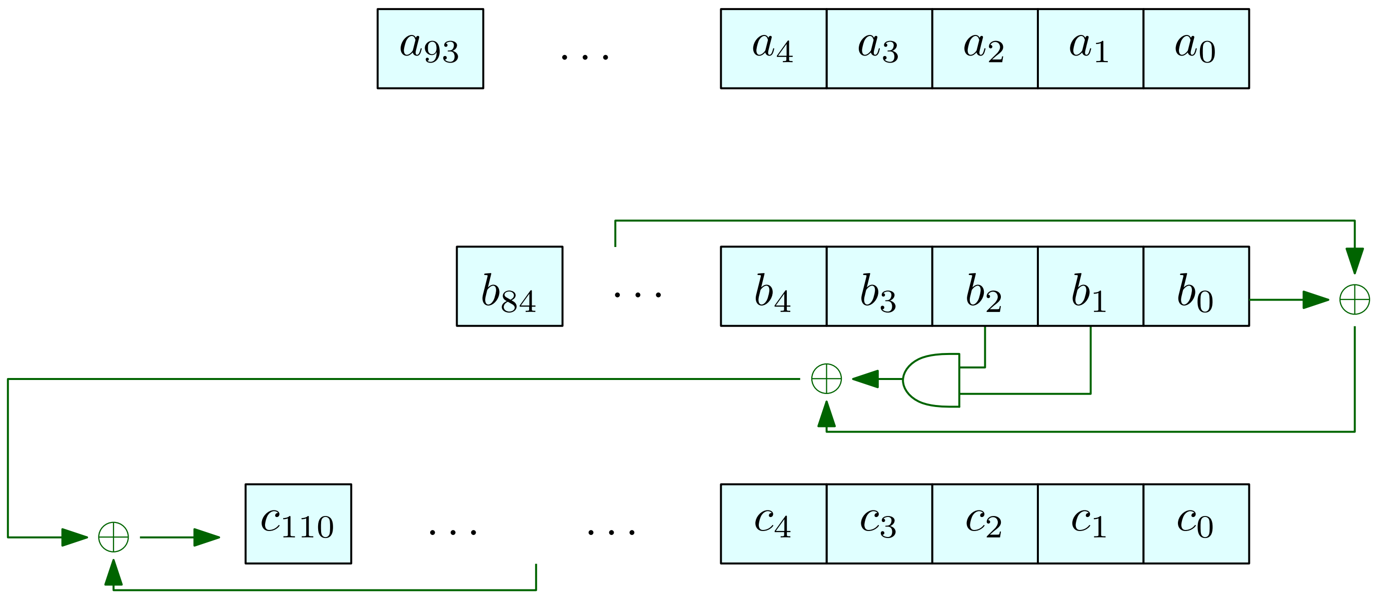
- Three FSRs (say A, B, C) of degrees 93, 84, and 111 (overall, the state is 288 bits long)



- The FSRs are **coupled**: the input of each FSR is a non-linear function of a register from that FSR, and of 4 registers from another FSR

Back to Trivium

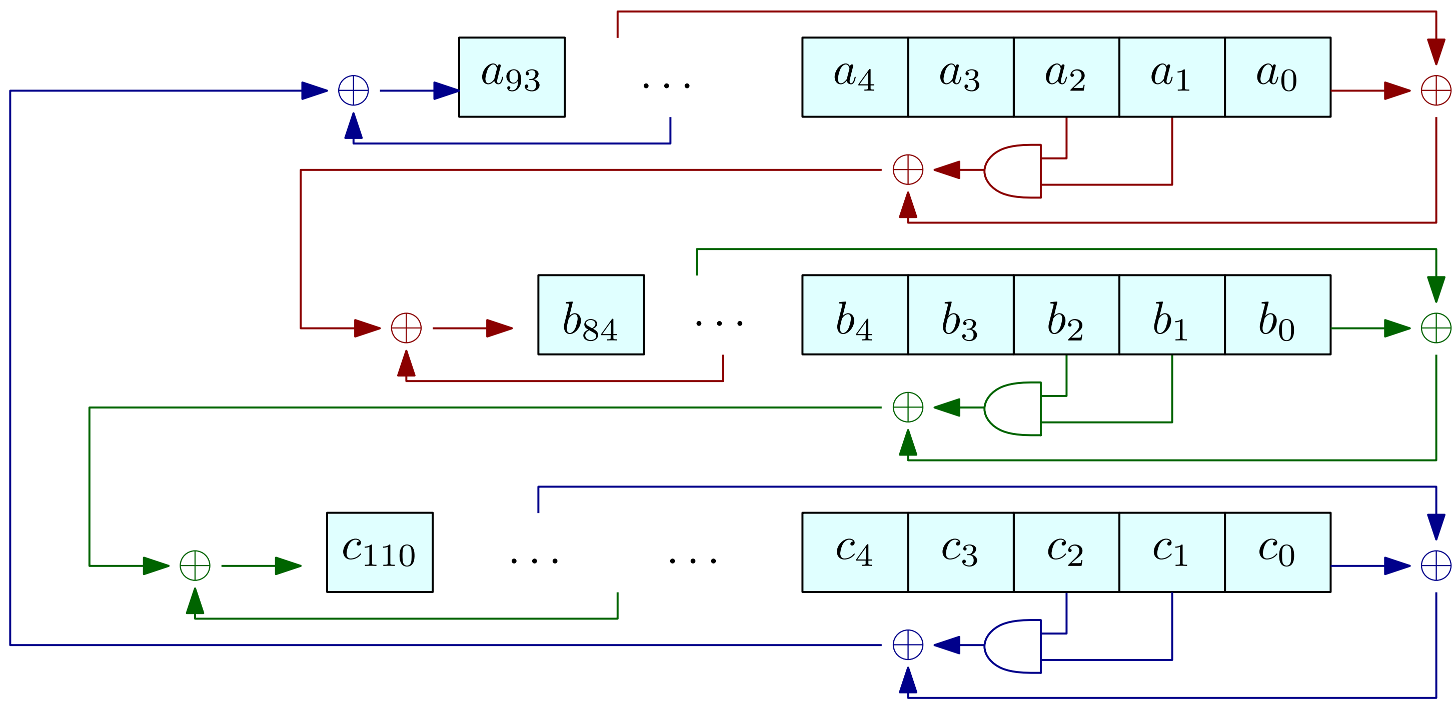
- Three FSRs (say A, B, C) of degrees 93, 84, and 111 (overall, the state is 288 bits long)



- The FSRs are **coupled**: the input of each FSR is a non-linear function of a register from that FSR, and of 4 registers from another FSR

Back to Trivium

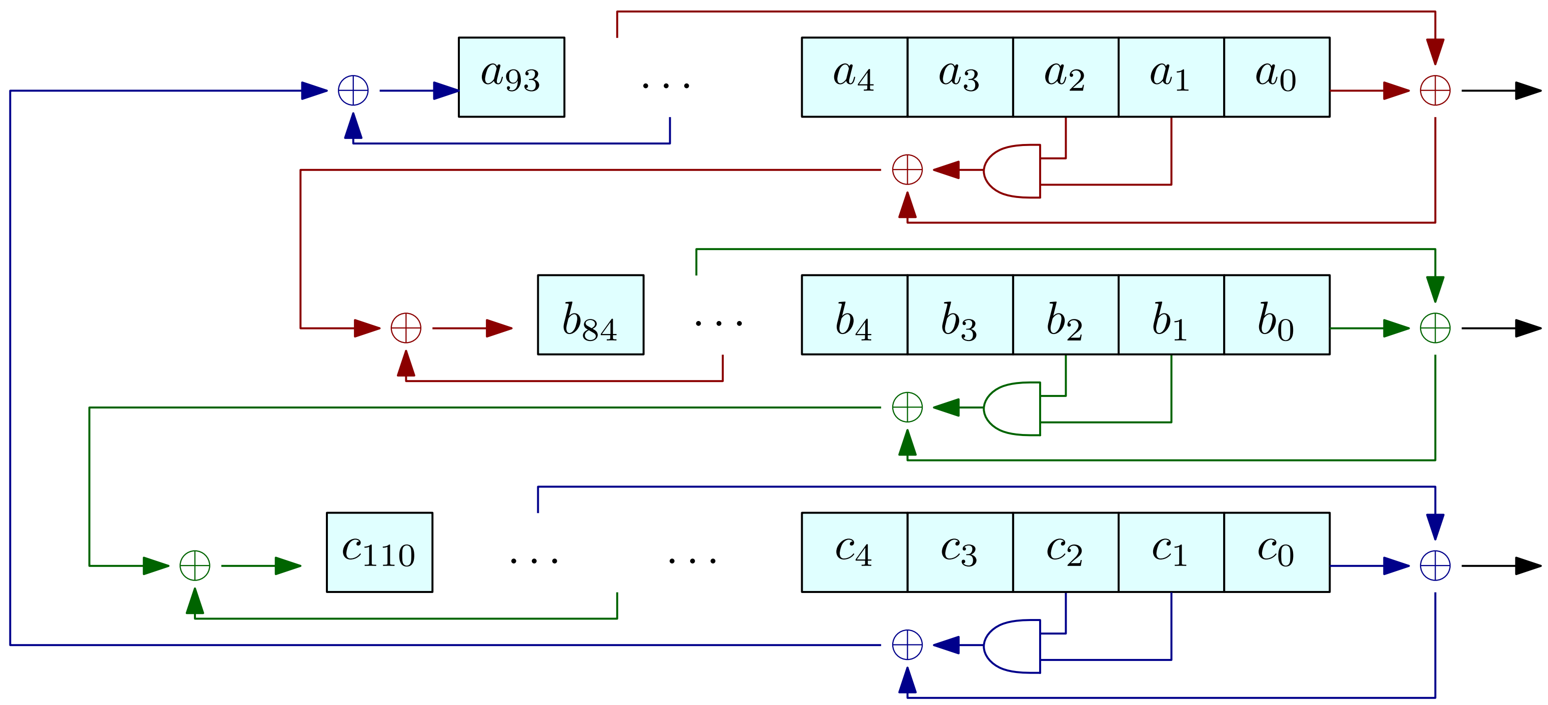
- Three FSRs (say A, B, C) of degrees 93, 84, and 111 (overall, the state is 288 bits long)



- The FSRs are **coupled**: the input of each FSR is a non-linear function of a register from that FSR, and of 4 registers from another FSR

Back to Trivium

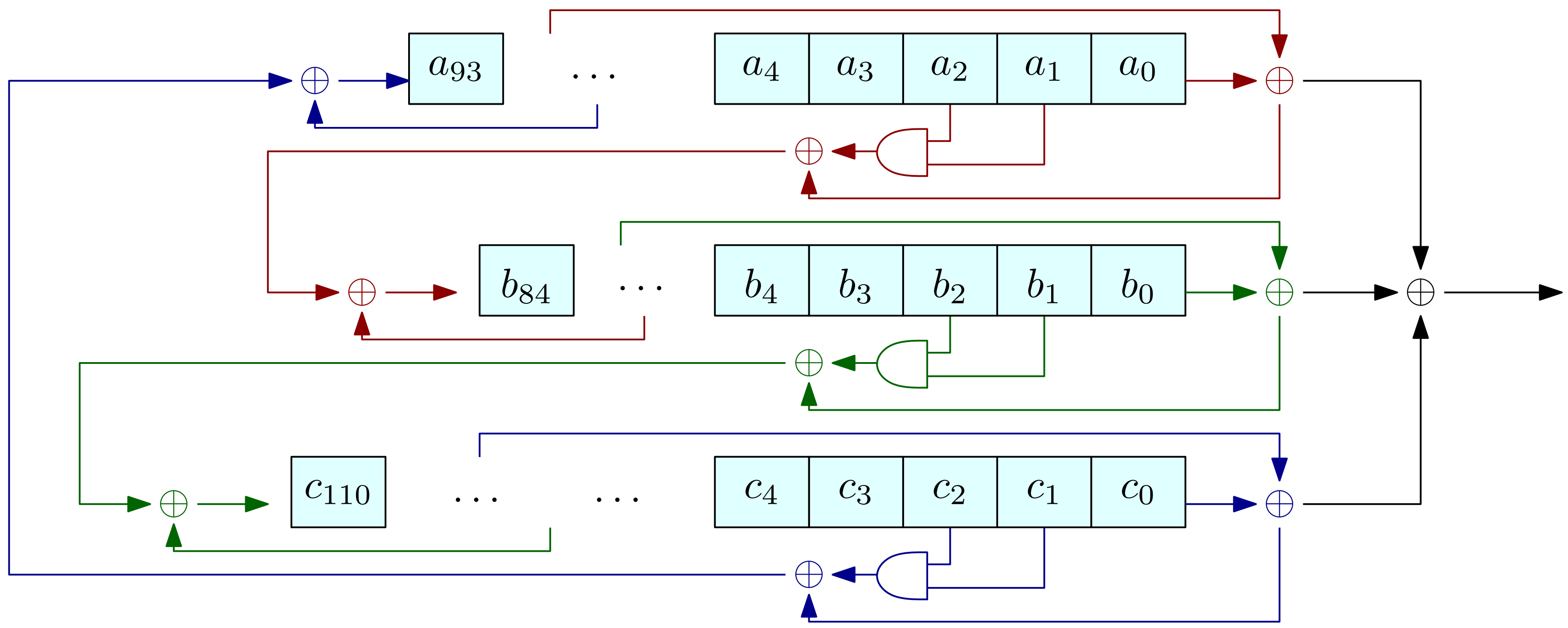
- Three FSRs (say A, B, C) of degrees 93, 84, and 111 (overall, the state is 288 bits long)



- The output of each FSR is the XOR of its rightmost bit plus the content of another register

Back to Trivium

- Three FSRs (say A, B, C) of degrees 93, 84, and 111 (overall, the state is 288 bits long)



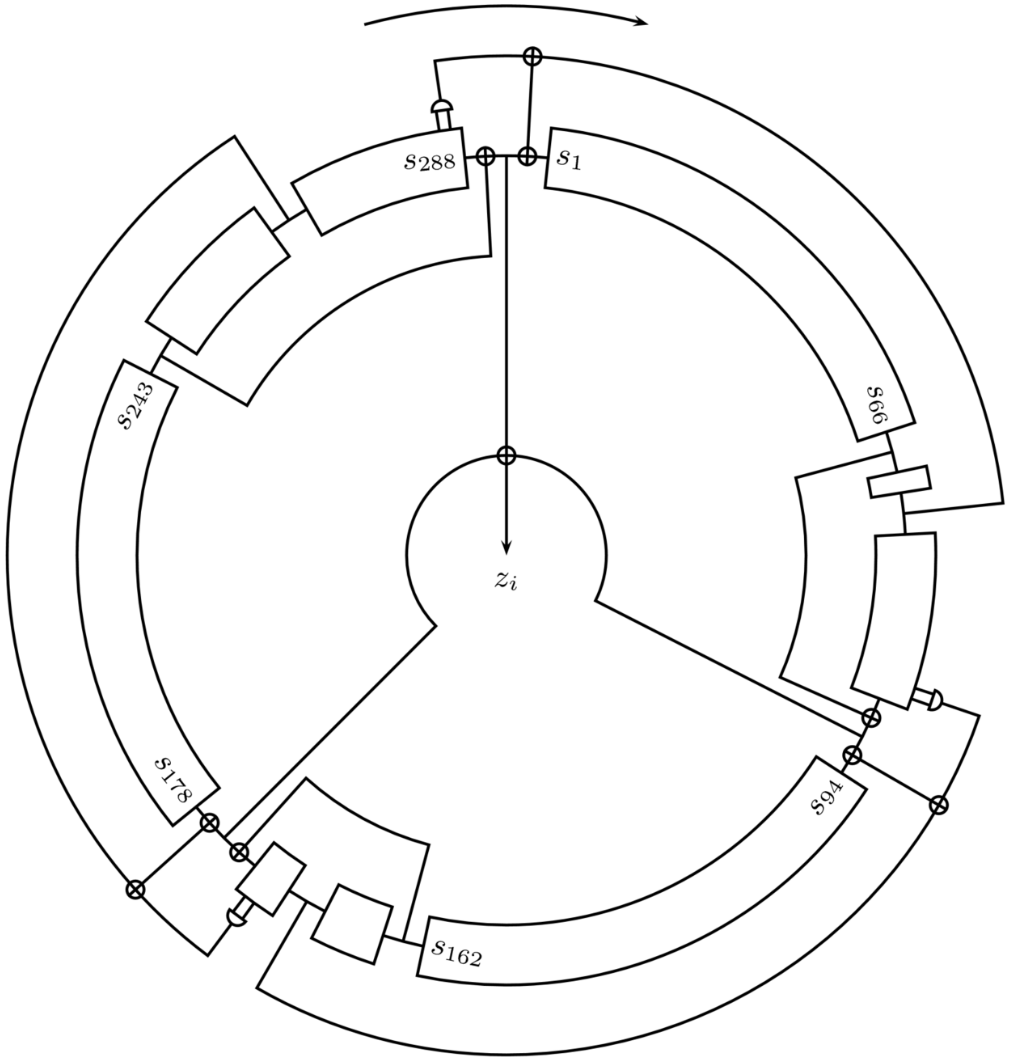
- The output of each FSR is the XOR of its rightmost bit plus the content of another register
- The output of Trivium is the XOR of the outputs of the single FSRs

Trivium: Init

Trivium takes a 80-bit key and a 80-bit IV... and generates up to 2^{64} bits of output

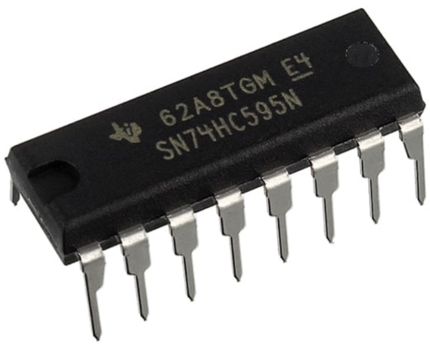
Init:

- Set the leftmost 80 registers of A to the key, and other registers to 0
- Set the leftmost 80 registers of B to the IV, and other registers to 0
- Set the rightmost 3 registers of C to 1, and other registers to 0
- Run for $4 \cdot 288$ clock ticks and discard the output


s_{288}
s_1
s_{99}
s_{243}
z_i
s_{178}
s_{94}
s_{162}

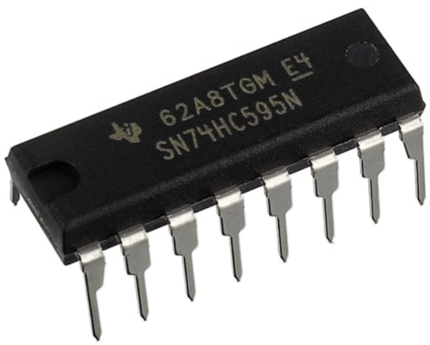
Extra: Implementing LFSRs in hardware

8 bit shift register



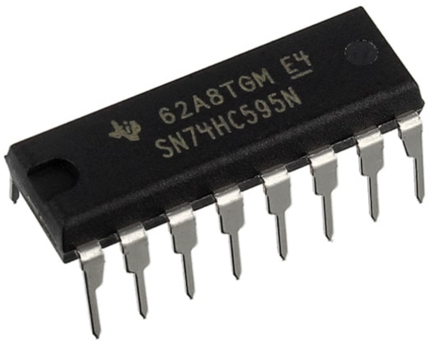
Extra: Implementing LFSRs in hardware

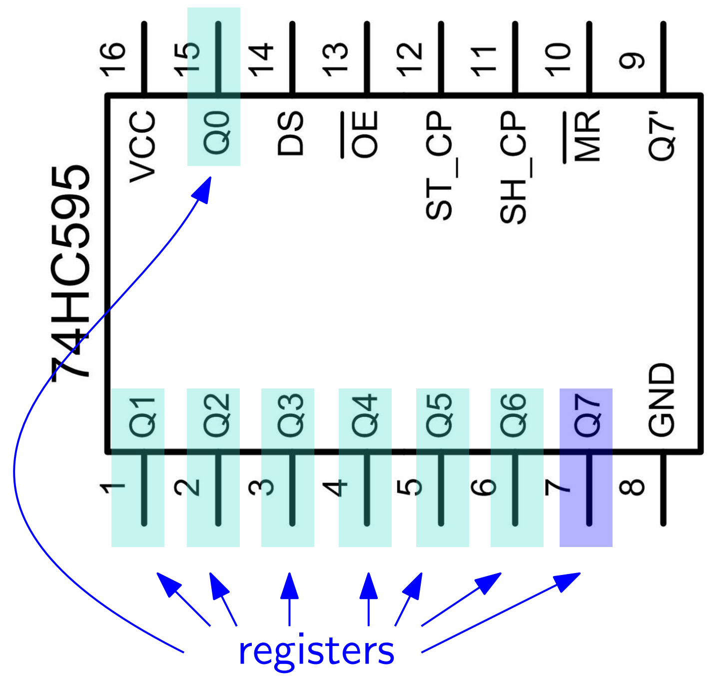
8 bit shift register



Extra: Implementing LFSRs in hardware

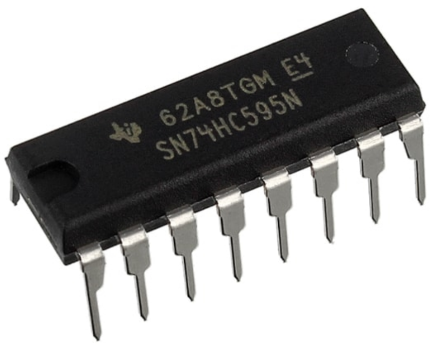
8 bit shift register

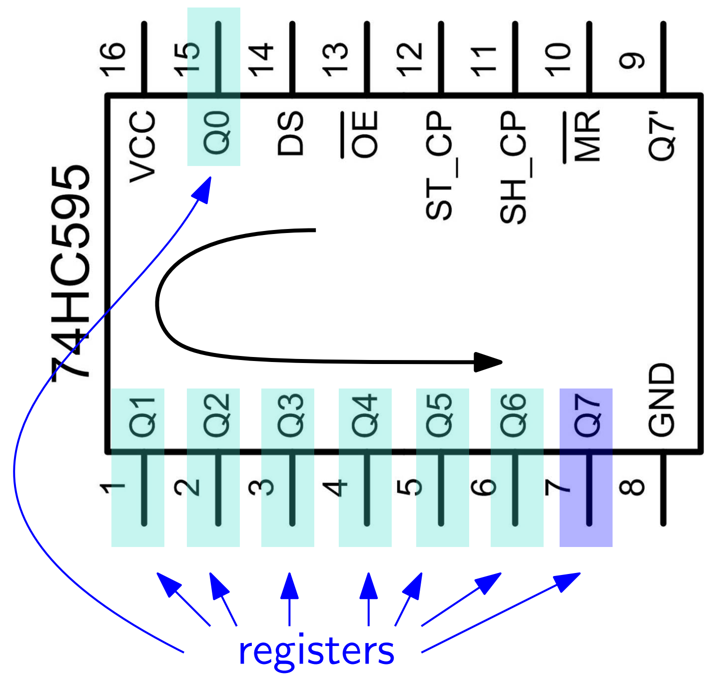




Extra: Implementing LFSRs in hardware

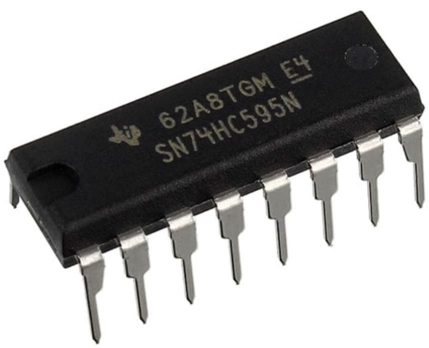
8 bit shift register

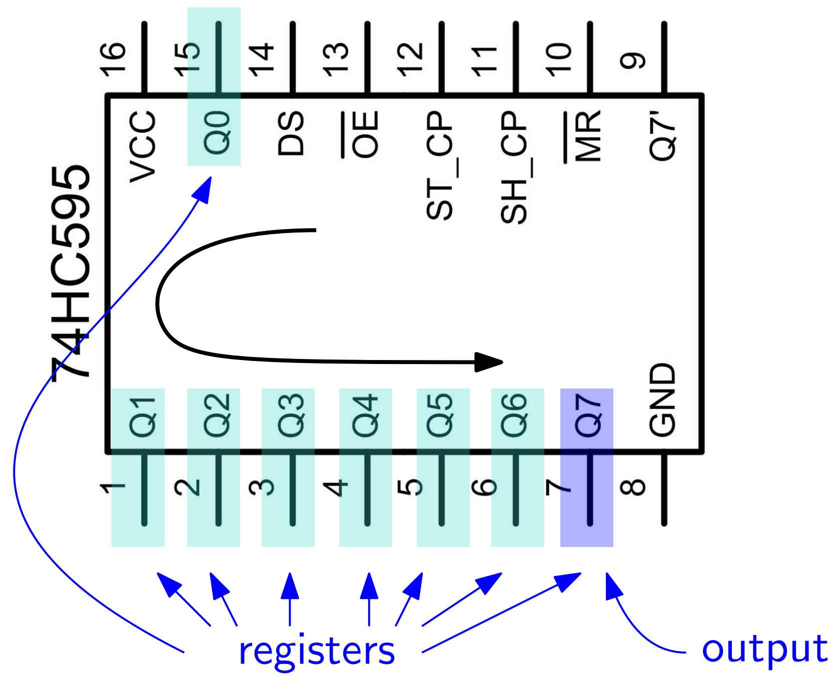




Extra: Implementing LFSRs in hardware

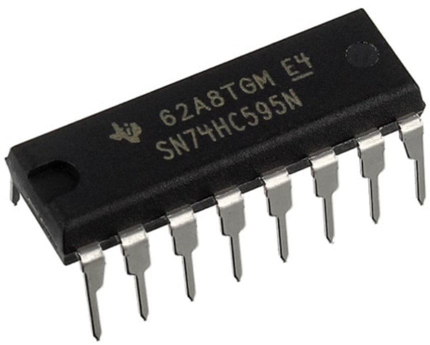
8 bit shift register

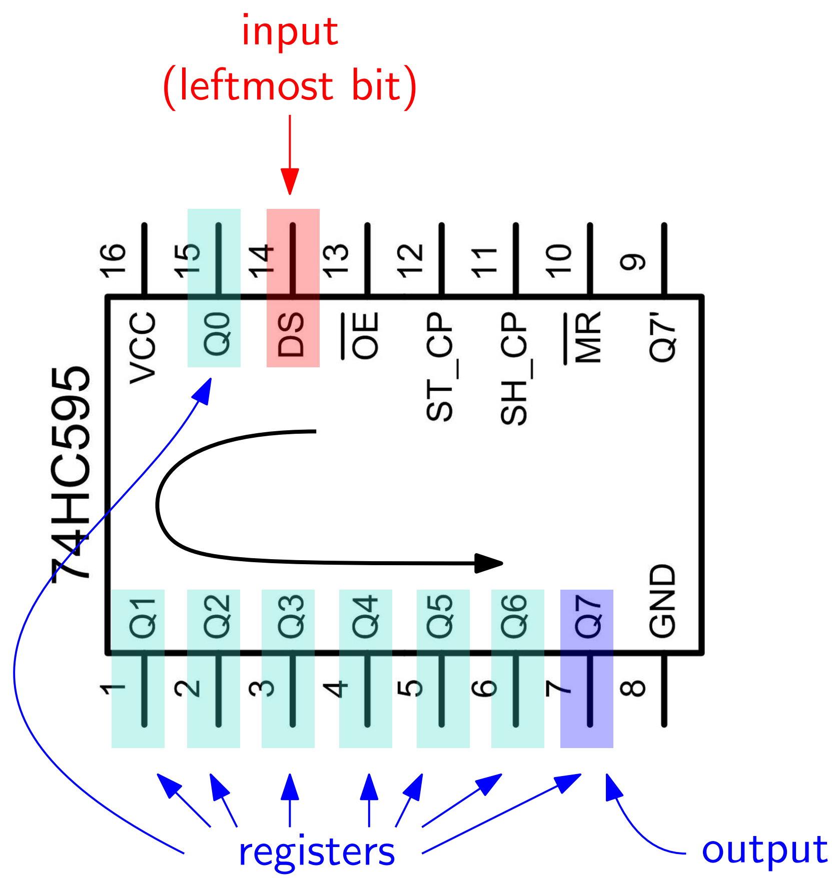




Extra: Implementing LFSRs in hardware

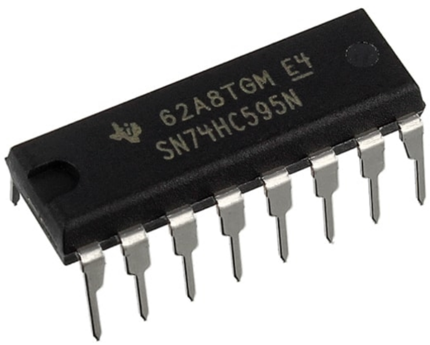
8 bit shift register

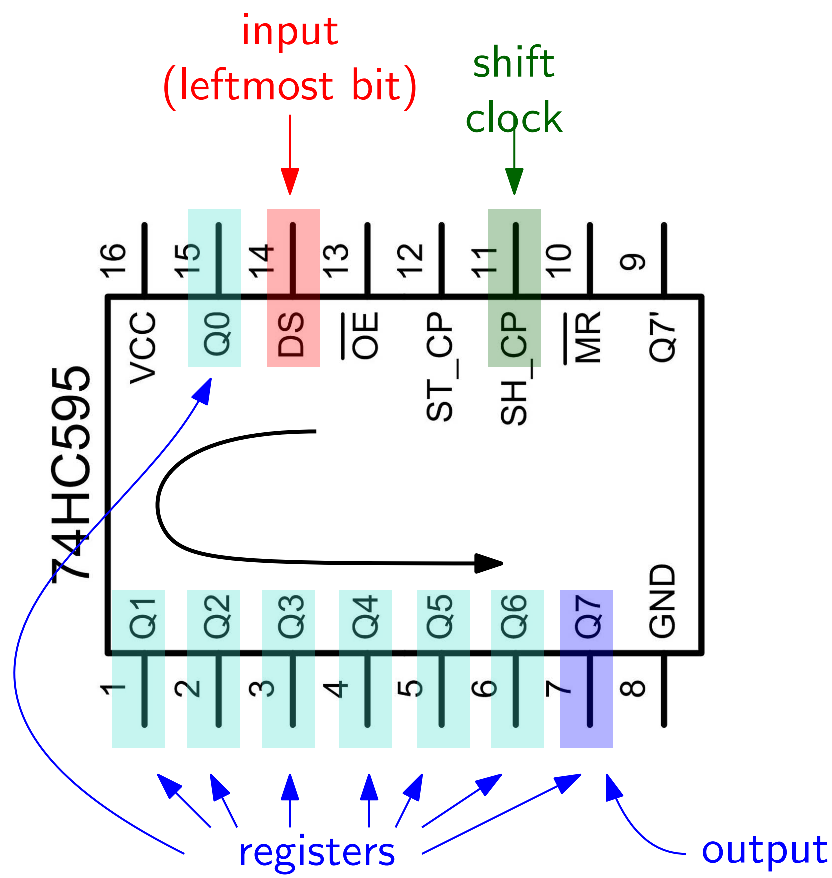




Extra: Implementing LFSRs in hardware

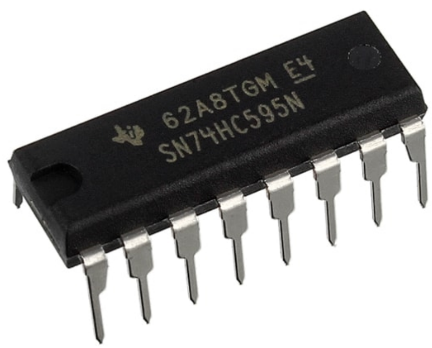
8 bit shift register

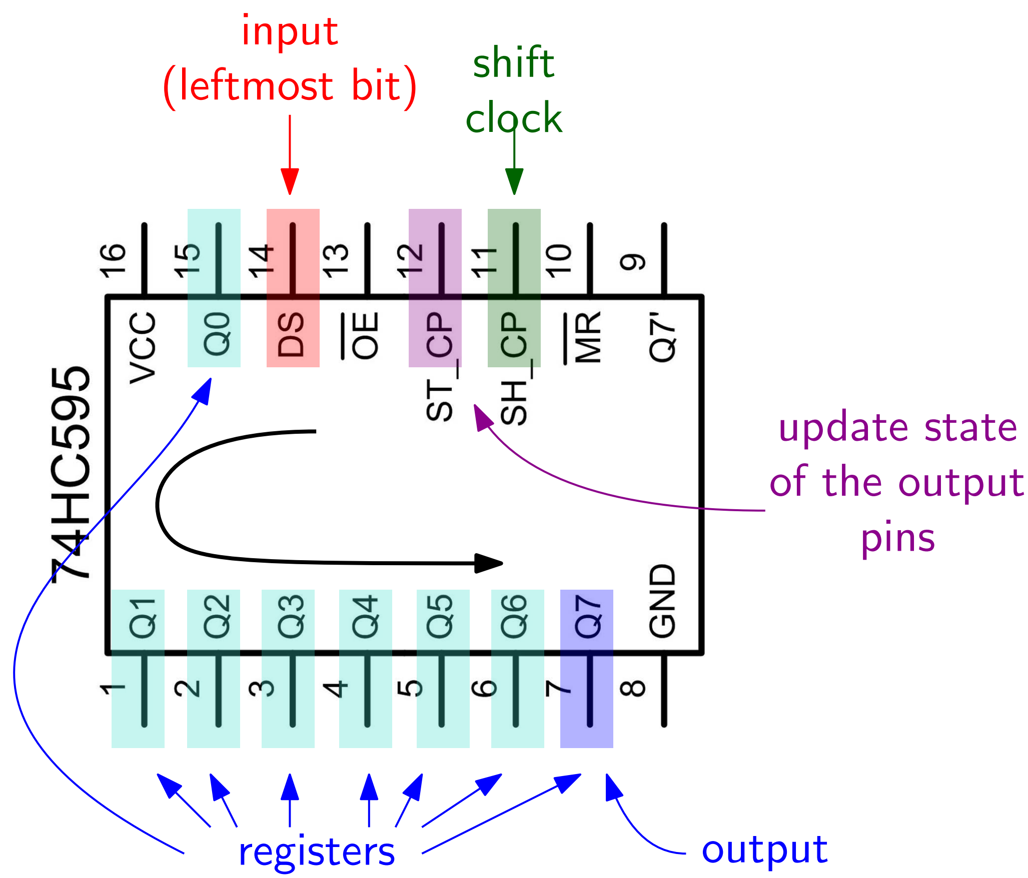




Extra: Implementing LFSRs in hardware

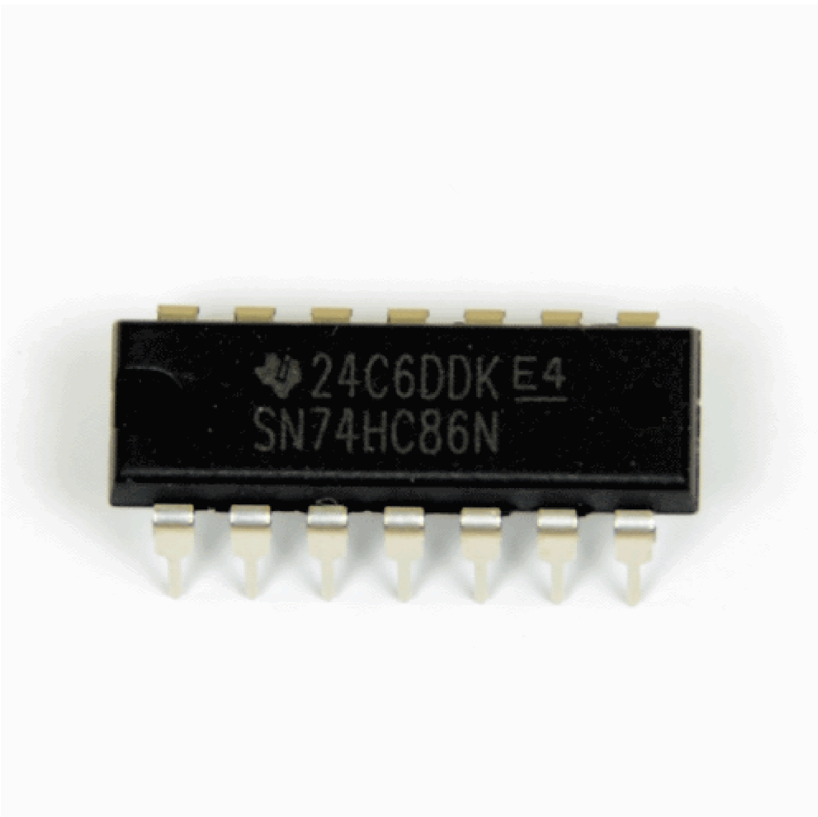
8 bit shift register

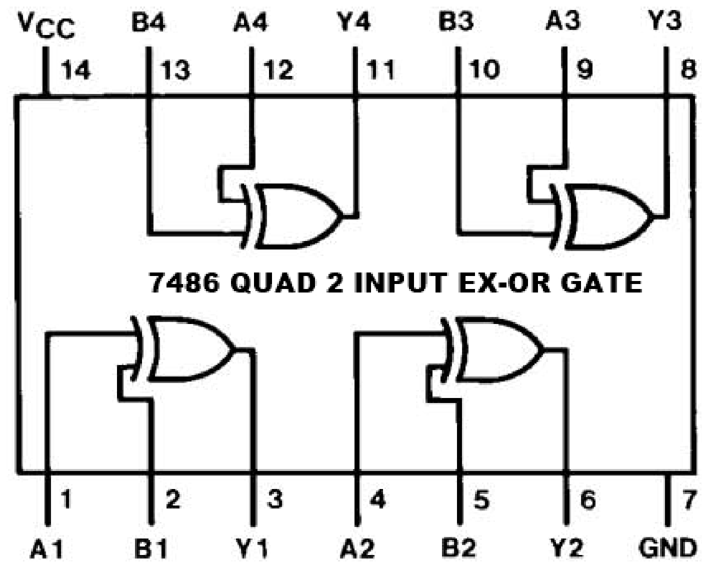




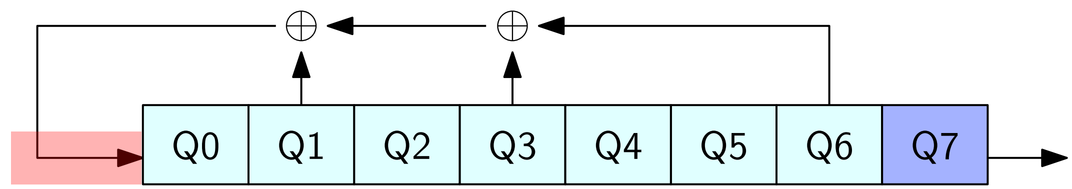
Extra: Implementing LFSRs in hardware

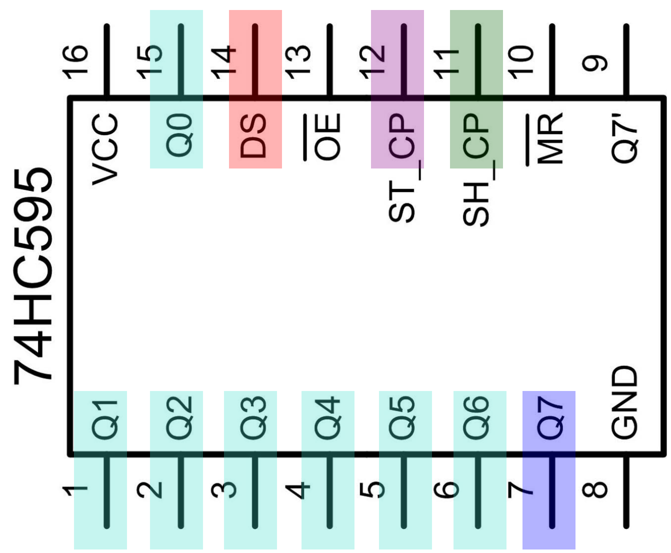
XOR gates

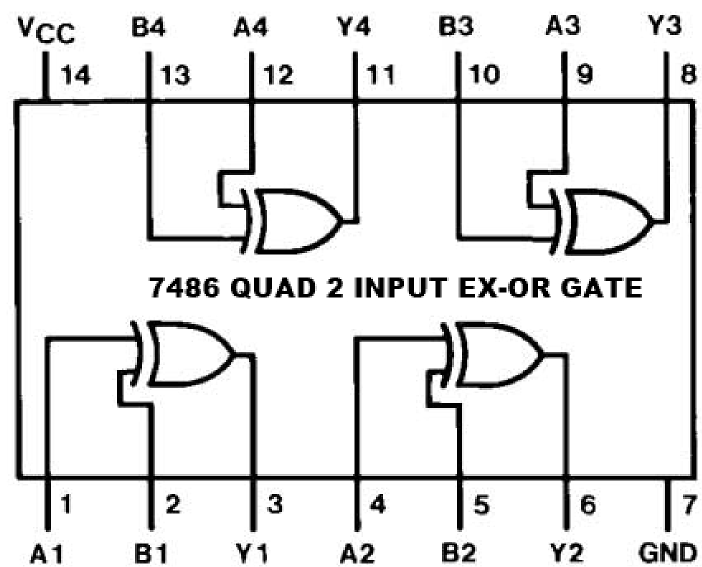




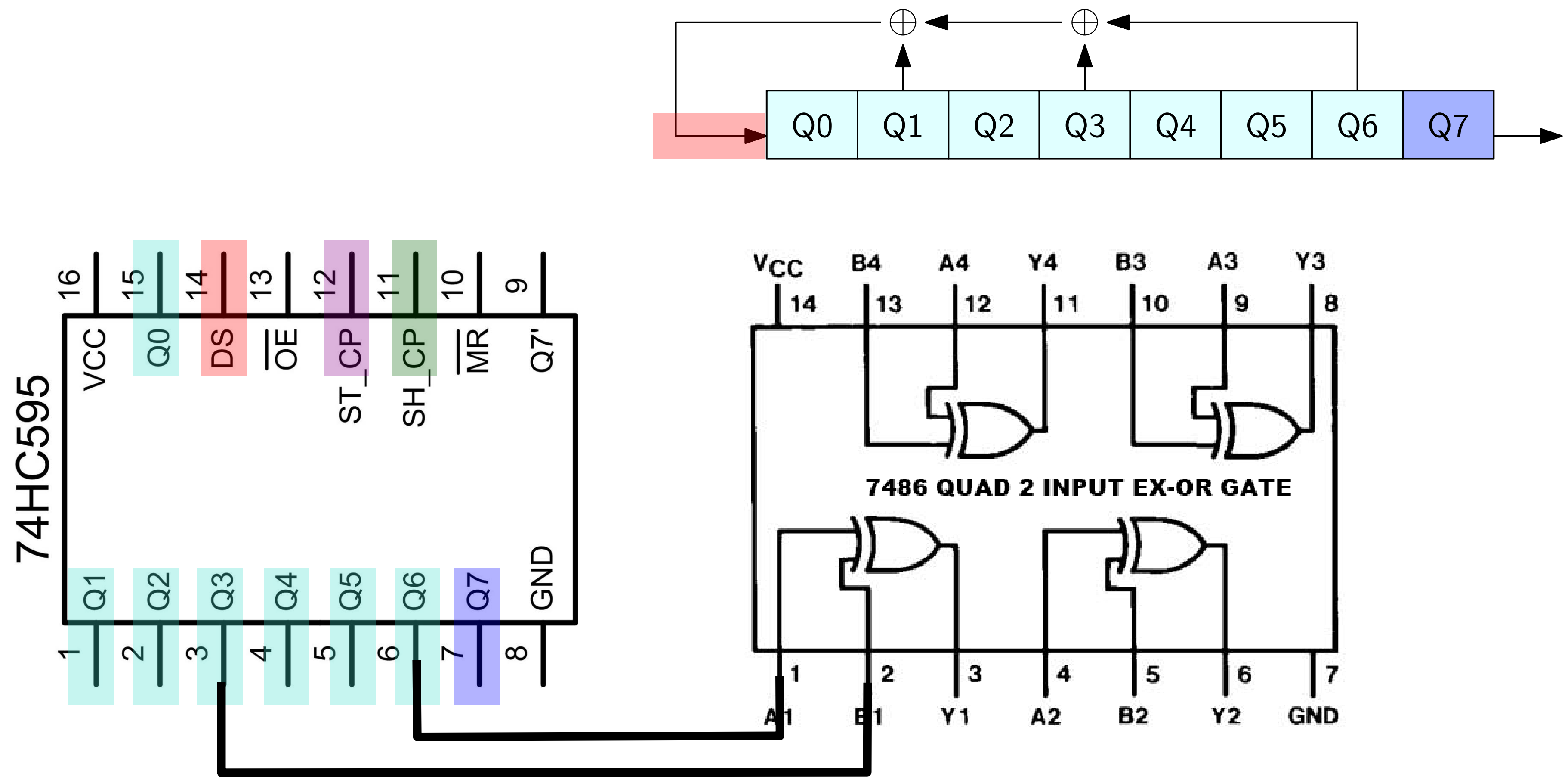
Extra: Implementing LFSRs in hardware



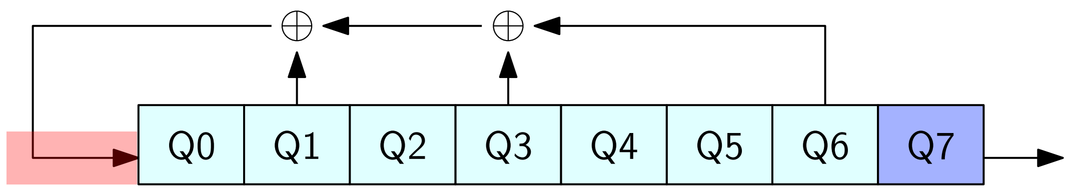


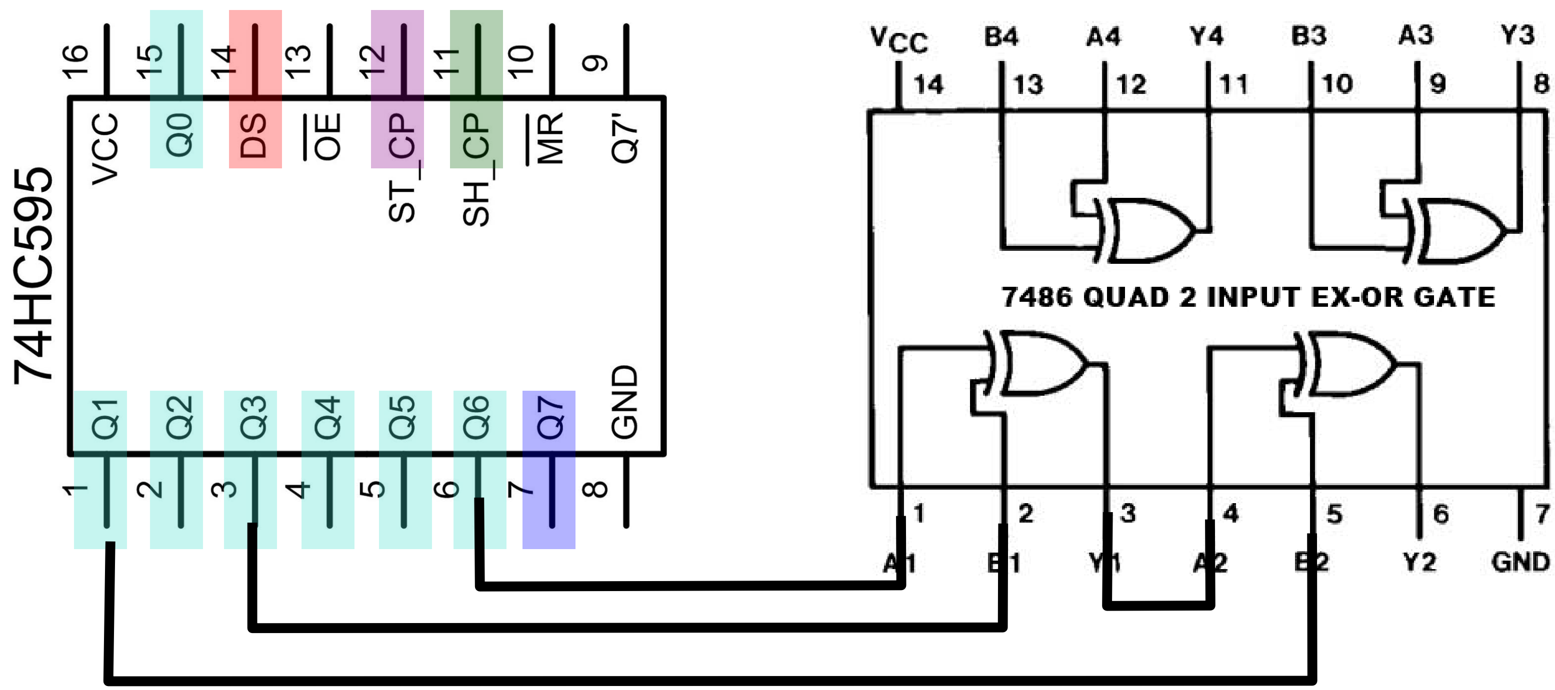


Extra: Implementing LFSRs in hardware

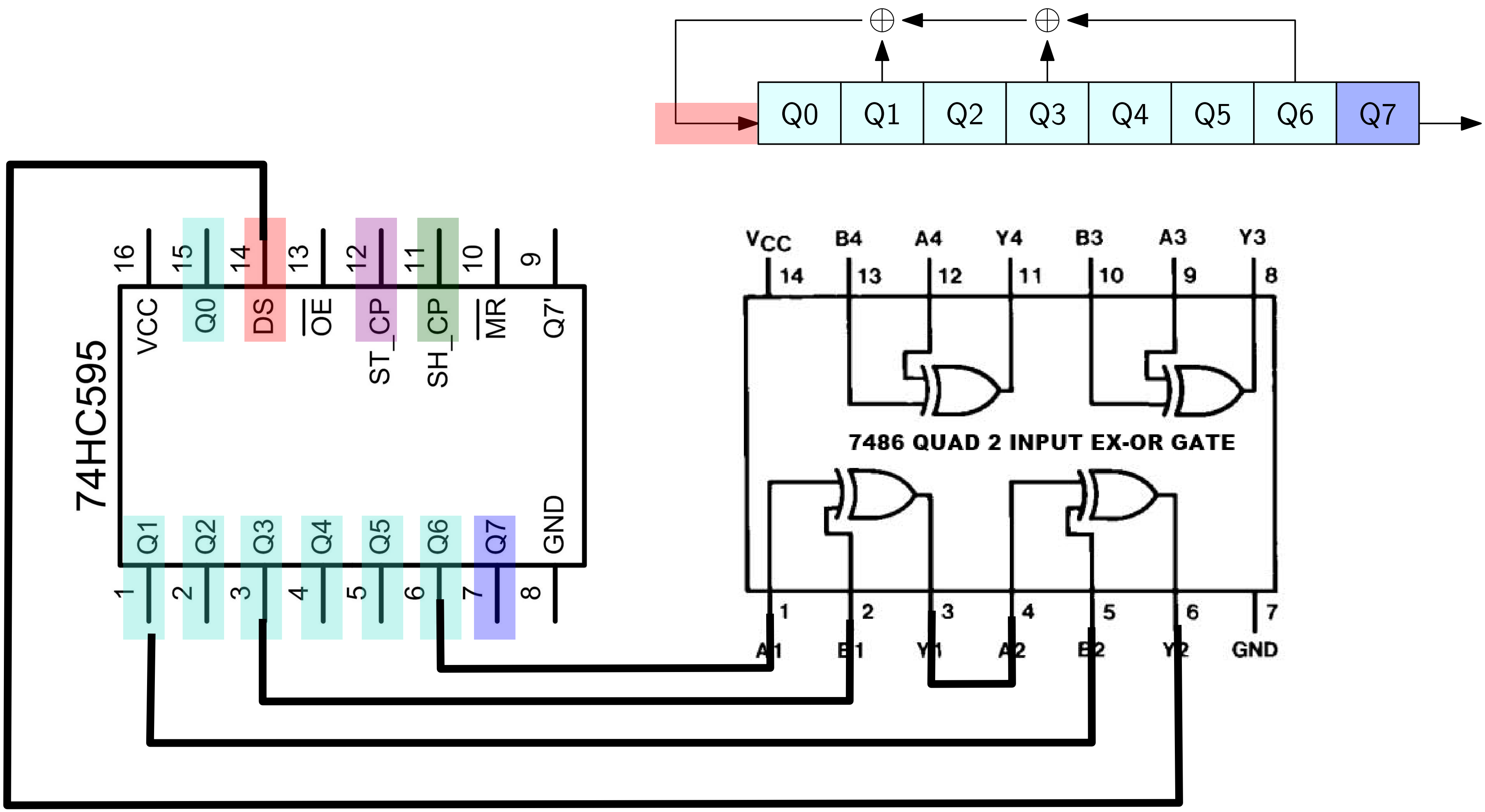


Extra: Implementing LFSRs in hardware





Extra: Implementing LFSRs in hardware



Extra: Implementing LFSRs in hardware