An Improved Hardware Implementation of the Quark Hash Function

Shohreh Sharif Mansouri and Elena Dubrova
Department of Electronic Systems
Royal Institute of Technology (KTH), Stockholm
Email:{shsm,dubrova}@kth.se

Overview

- Motivation
- Structure of the Quark hash function
- Techniques to improve implementation
- Experimental results
- Conclusion

The Main Goal

- Improving Quark in terms of **Throughput**, Area and Power

- We achieve it by modifying the architecture of Quark without changing its algorithm

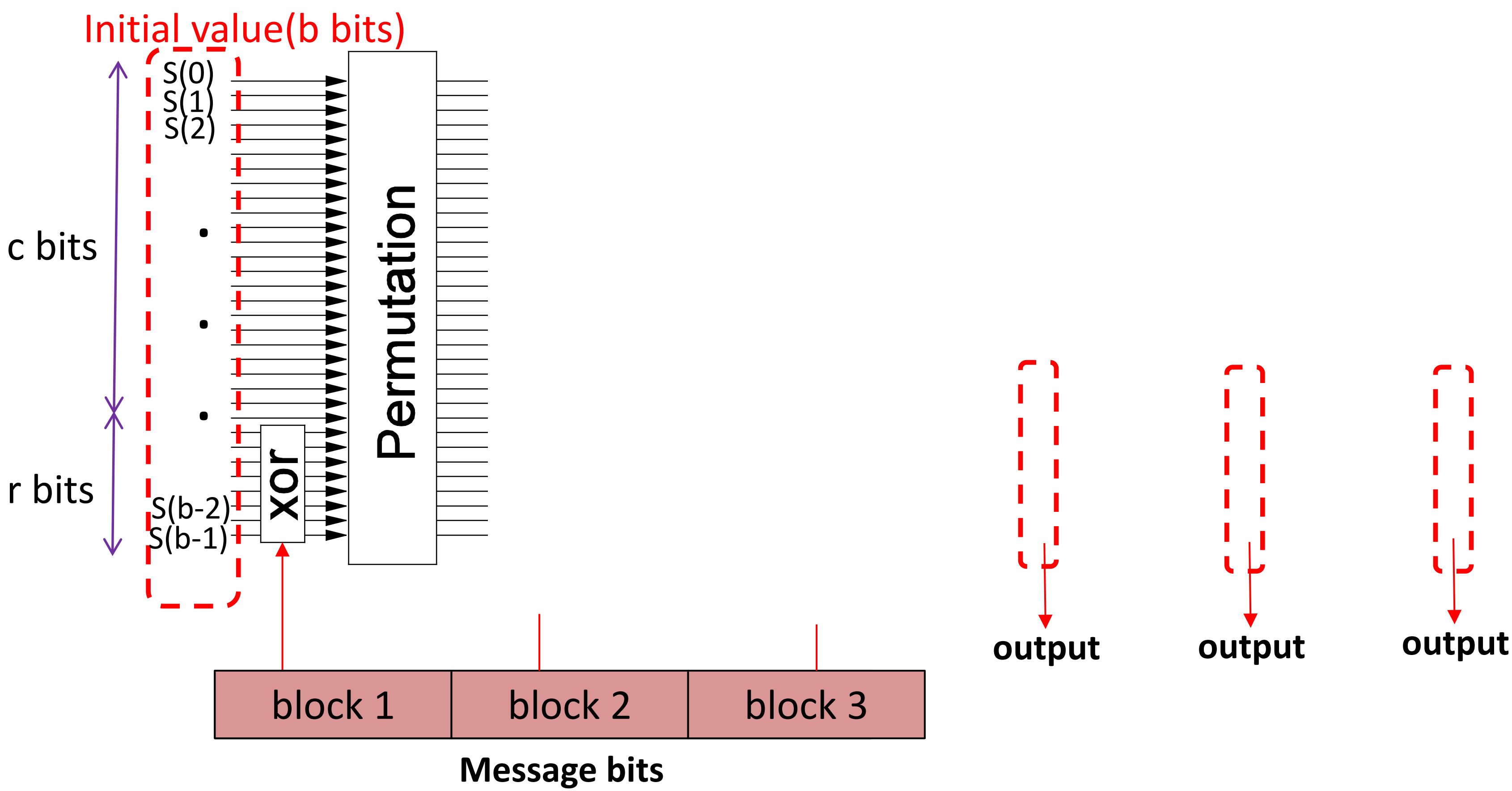
- We succeed to increase the throughput by 34% for U-Quark

Quark Family of Hash Function

- Quark is a family of cryptographic sponge functions
- Targets resource-constrained hardware environments
- Three Quark instances: U- Quark , D-Quark and S-Quark
- Supports at least 64-bits, 80-bits and 112-bits security level against most crypto-attacks.

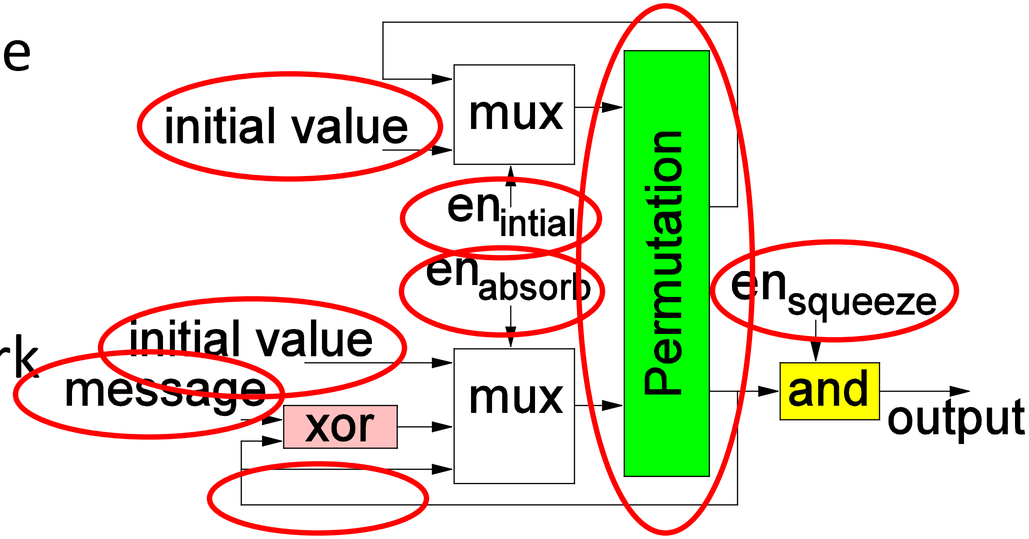
Sponge Construction

- A sponge construction goes through three phases:
  - Initialization
  - Absorbing phase
  - Squeezing phase



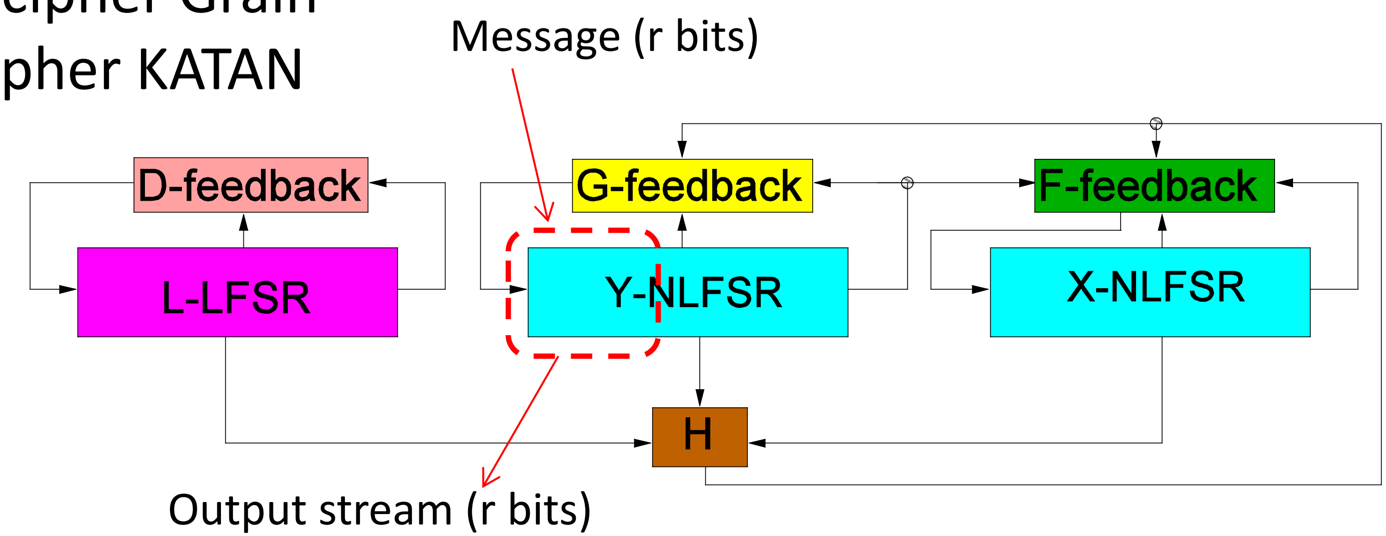
Quark Hardware Structure

- The sponge construction can be implemented serially, with a single permutation block.



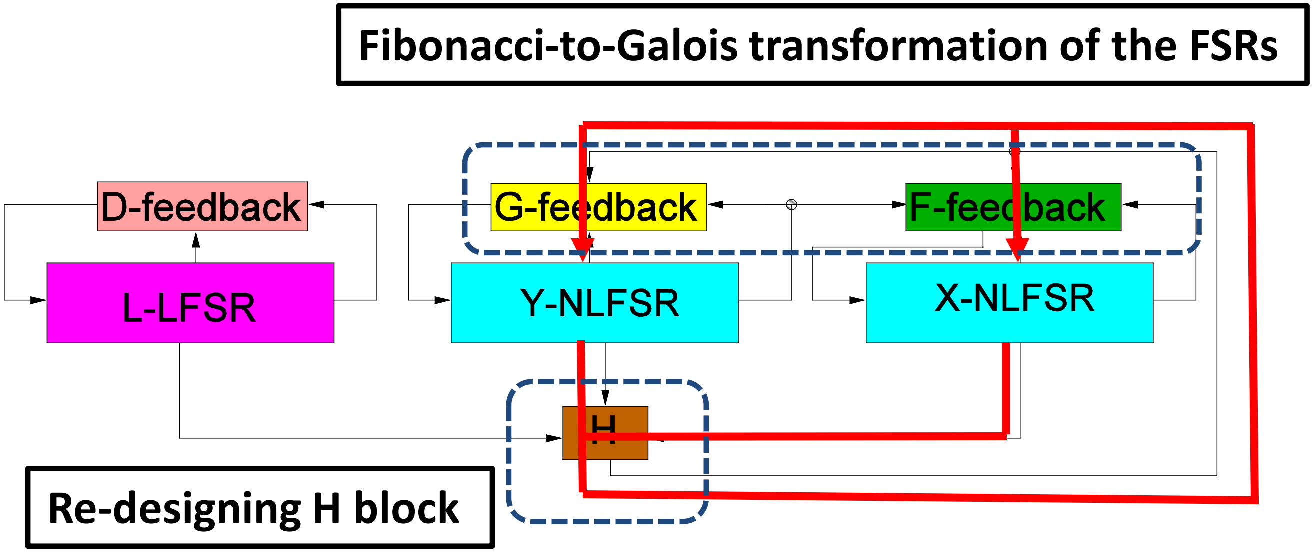
- The permutation block of Quark is based on shift registers

- It is inspired by:
  - stream cipher Grain
  - block cipher KATAN



How to Improve Throughput?

- Throughput is determined by the critical path, which is the longest combinational path in the system.
- Quark 's critical:
  - **Dhn**: maximal delay from a flip-flop of one of the NLFSRs through the **h** functions to the first flip-flop of one of the NLFSRs

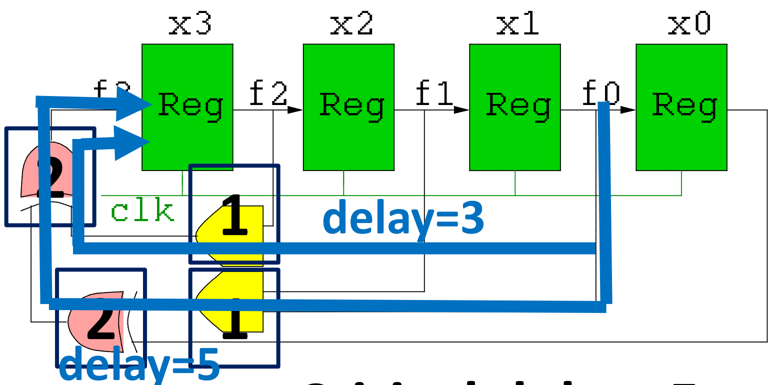


Fibonacci to Galois Transformation

- Improves the critical path delay
- Brings no area or power penalty

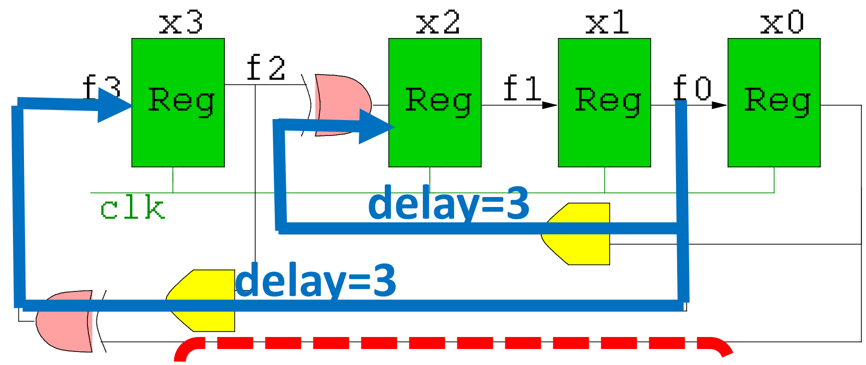
Fibonacci to Galois Transformation*

Fibonacci Configuration



Critical delay=5

Galois Configuration



Critical delay=3

f3=x0 + x1x3 +x1x2
f2=x3
f1=x2
f0=x1

f3=x0 + x1x3
f2=x3 +x0x1
f1=x2
f0=x1

*A Transformation from the Fibonacci to the Galois NLFSRs", E. Dubrova,IEEE *Transactions on Information Theory*, 55:11, 2009, pp. 5263-5271

Example

The transformation from Fibonacci to Galois is not unique

$$\begin{aligned} f_3 &= x_1x_2 + \boxed{x_1x_3} + x_0 \\ f_2 &= x_3 \\ f_1 &= x_2 \\ f_0 &= x_1 \end{aligned} \qquad \begin{aligned} f_3 &= \boxed{x_1x_2} + x_0 \\ f_2 &= x_3 + \boxed{x_0x_2} \\ f_1 &= x_2 \\ f_0 &= x_1 \end{aligned} \qquad \begin{aligned} f_3 &= x_0 \\ f_2 &= x_3 + \boxed{x_0x_1} + x_0x_2 \\ f_1 &= x_2 \\ f_0 &= x_1 \end{aligned}$$

Fibonacci to Galois Transformation

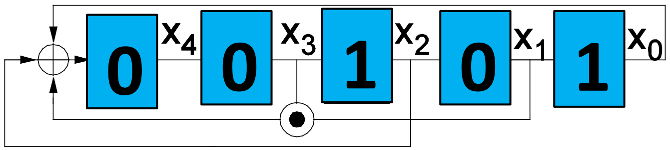
- Explore the design space to find the best Galois NLFSR equivalent to a given Fibonacci NLFSR
- Optimal algorithm: synthesize every possible combination and find the best solution

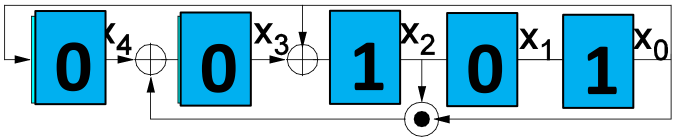
  Computationally unfeasible - we need a heuristic approach* **F2G:*http://web.it.kth.se/~dubrova/fib2gal.html***

*"An Algorithm for Constructing a Fastest Galois NLFSR Generating a Given Sequence", J.-M.,Chabloz, S. Mansouri, E. Dubrova, *in Sequences and Their Applications* , LNCS 6338, 2010, pp. 41-55

Loading

- Sometimes, with the same initial values, Fibonacci and Galois FSRs may produce different output streams.





$f_4 = x_0 + x_2 + x_1 x_3$

$f_3 = x_4$

$f_2 = x_3$

$f_1 = x_2$

$f_0 = x_1$

$f_4 = x_0$

$f_3 = x_4 + x_0 x_2$

$f_2 = x_3 + x_0$

$f_1 = x_2$

$f_0 = x_1$

Not same output stream

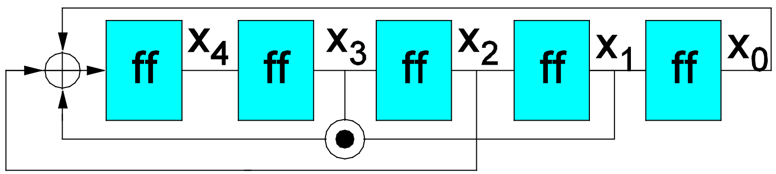
cycle	x_4	x_3	x_2	x_1	x_0
1	0	0	1	0	1

cycle	x_4	x_3	x_2	x_1	x_0
1	0	0	1	0	1

Loading

- The Fibonacci FSR and the Galois FSR are loaded in parallel with the same value

- Update functions of the Galois FSR are "turned on" one by one



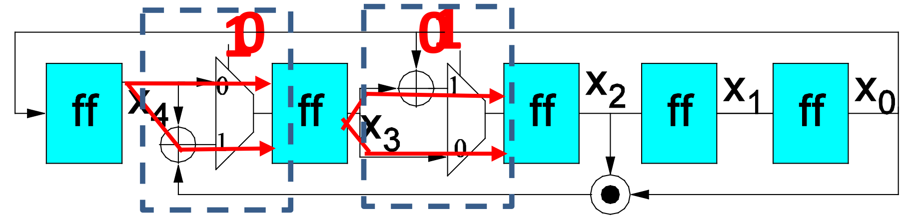
$f_4 = x_0 + x_2 + x_1 x_3$

$f_3 = x_4$

$f_2 = x_3$

$f_1 = x_2$

$f_0 = x_1$



$f_4 = x_0$

$f_3 = x_4 + x_0 x_2$

$f_2 = x_3 + x_0$

$f_1 = x_2$

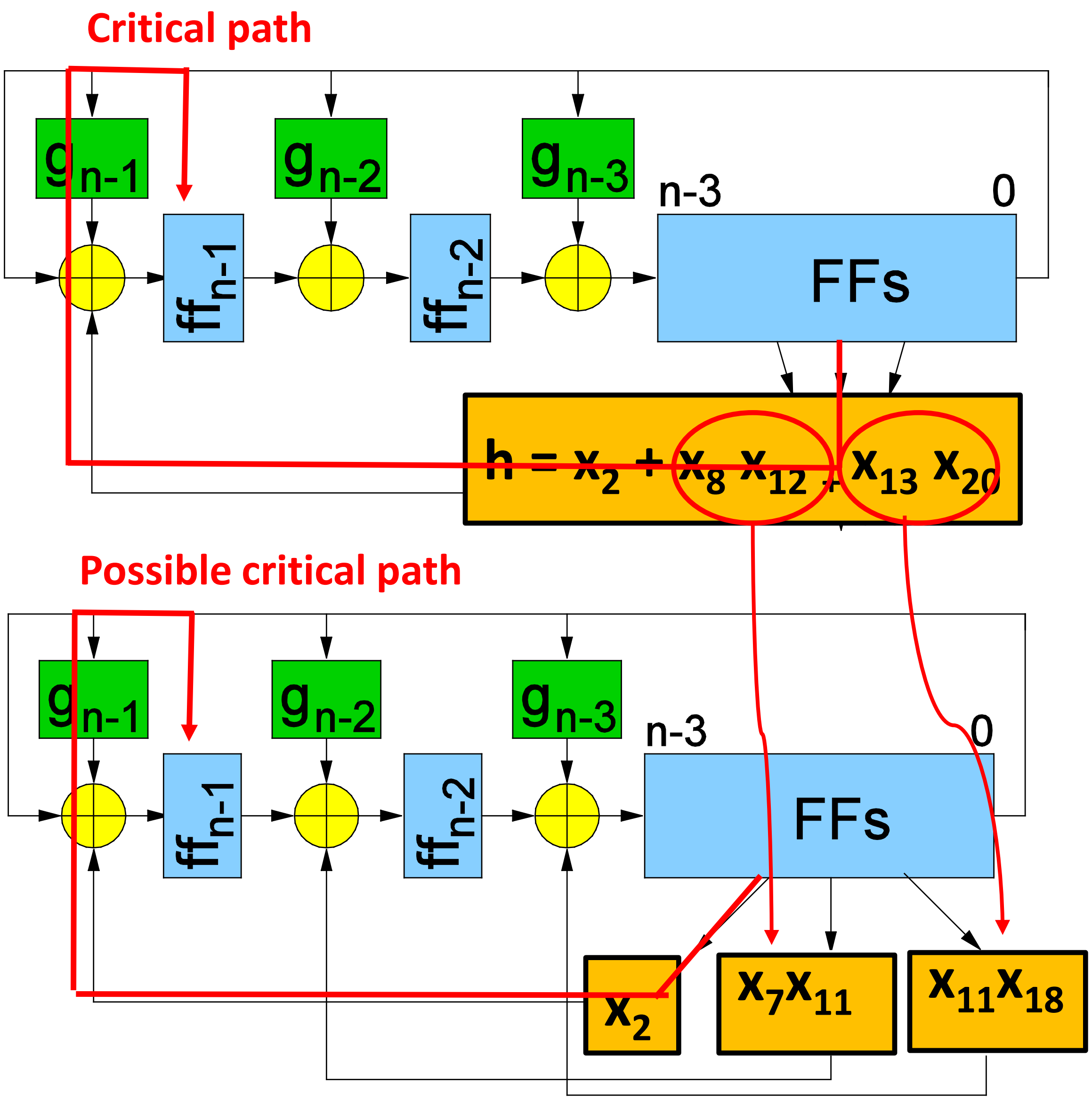
$f_0 = x_1$

same output stream

cycle	x_4	x_3	x_2	x_1	x_0
1	0	0	1	0	1

cycle	x_4	x_3	x_2	x_1	x_0
1	0	0	1	0	1

Re-designing the Filter Generator



$x_{n-1} = x_0 + g_{n-1} + h$

$x_{n-2} = x_{n-1} + g_{n-2}$

$x_{n-3} = x_{n-2} + g_{n-3}$

$x_{n-4} = x_{n-3}$

…

…

$x_0 = x_1$

$x_{n-1} = x_0 + g_{n-1} + h_{n-1}$

$x_{n-2} = x_{n-1} + g_{n-2} + h_{n-2}$

$x_{n-3} = x_{n-2} + g_{n-3} + h_{n-3}$

$x_{n-4} = x_{n-3}$

…

…

$x_0 = x_1$

Implementation Results for U-Quark

- Throughput improvement: 34%
- Power improvement: 15%
- Area overhead is less than 1%

Other Achieved Improvements

- We improved the hardware implementation of some FSR based stream cipher.
- The best achieved improvements are for Grain-80, Grain-128 and Grain-128a.

	Grain-128a*	Grain-128**	Grain-80**	Quark
Freq.	**52%**	**47%**	**42%**	**34%**
Area	**-5%**	**6%**	**5%**	**-1%**
Power	**2%**	**9%**	**11%**	**15%**

**"An Improved Hardware Implementation of the Grain Stream Cipher", S. Mansouri, E. Dubrova in Euromicro Conference on Digital System Design (DSD'2010)*

*** "An Improved Hardware Implementation of the Grain-128a Stream Cipher", S. Mansouri, E. Dubrova , in International Conference on Information Security and Cryptology (ICISC'2012)*

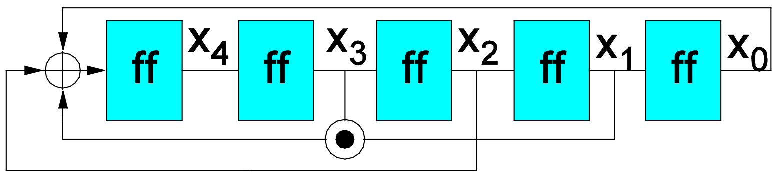
Conclusion

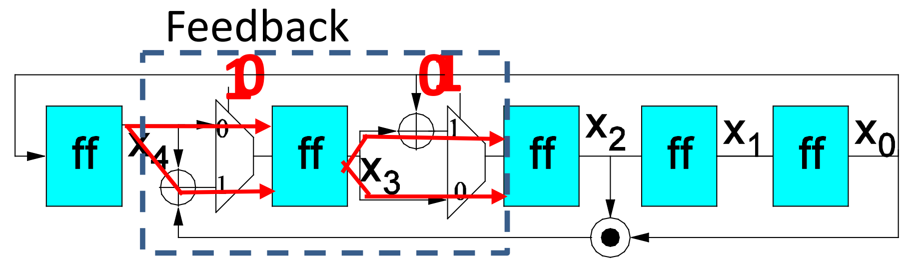
- High throughput improvement
- Limited area/power impact
- Techniques compatible with the standard ASIC flow
- Some techniques can be applied to other ciphers

Thank You for your attention

Questions?

F2G: *http://web.it.kth.se/~dubrova/fib2gal.html*





$$f_4 = x_0 + x_2 + x_1 x_3$$
$$f_3 = x_4$$
$$f_2 = x_3$$
$$f_1 = x_2$$
$$f_0 = x_1$$

$$f_4 = x_0$$
$$f_3 = x_4 + x_0 x_2$$
$$f_2 = x_3 + x_0$$
$$f_1 = x_2$$
$$f_0 = x_1$$

Start wth different initial value

same output stream

cycle	x_4	x_3	x_2	x_1	x_0
1	0	0	1	0	1

cycle	x_4	x_3	x_2	x_1	x_0
1	0	0	1	0	1