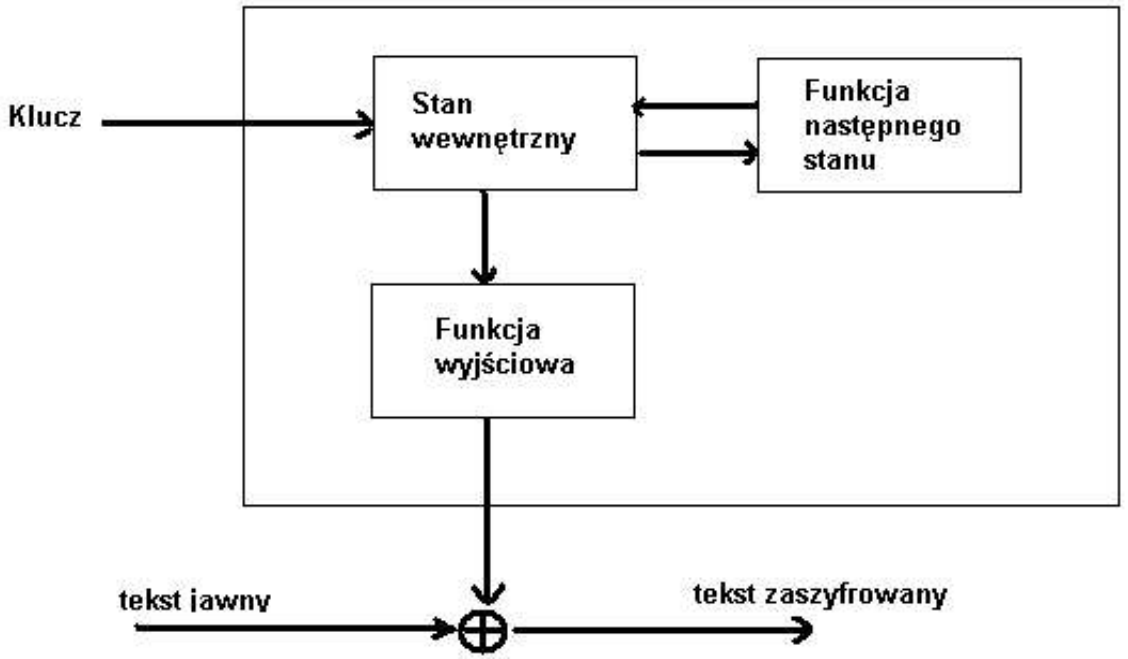
Sprzętowo-zorientowane konstrukcje szyfrów strumieniowych

Marcin Rogawski
rogawskim@prokom.pl

Plan referatu

- Szyfry strumieniowe,
- Zastosowanie i wymagania szyfrów strumieniowych ,
- Projekt eSTREAM – profil sprzętowy i programowy,
- A5, ..., Grain,Trivium ...

Szyfry strumieniowe



Czy szyfry strumieniowe są potrzebne ?

- Steve Babbage (Vodafone):
- Szyfr blokowy + tryb strumieniowy = szyfr strumieniowy,
- Standard FIPS197,
- AES – bardzo elastyczne implementacje sprzętowe i programowe,
- AES – uznane bezpieczeństwo (?),

Środowiska ekstremalne ;)

- Bardzo duże przepustowości,
- Bardzo mały pobór mocy,
- Bardzo efektywna implementacja,
- Elastyczność – 32, 64 bitowe procesory, 8 bitowe mikrokontrolery, FPGA, ASIC,

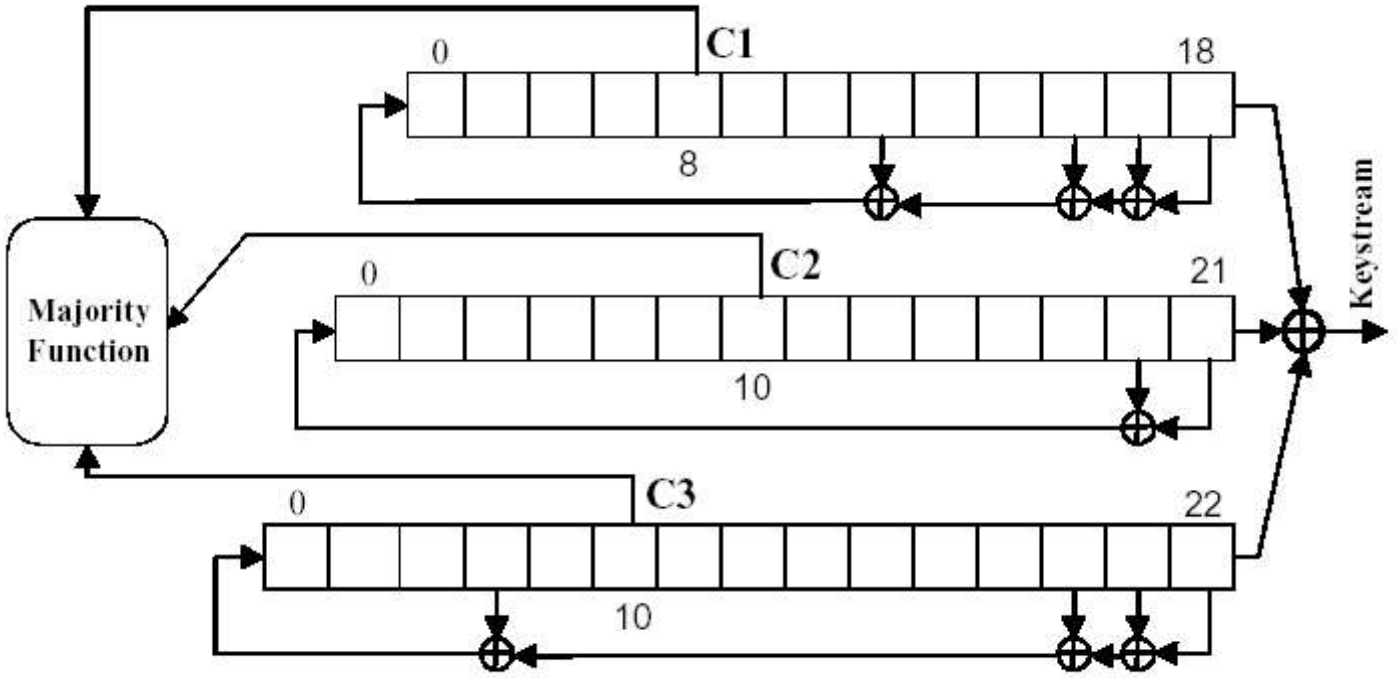
eSTREAM

- http://www.ecrypt.eu.org/stream/
- Projekt 2004-2008,
- Dwa profile algorytmów – sprzętowe i programowe,
- Faza I – III 2006,

eSTREAM

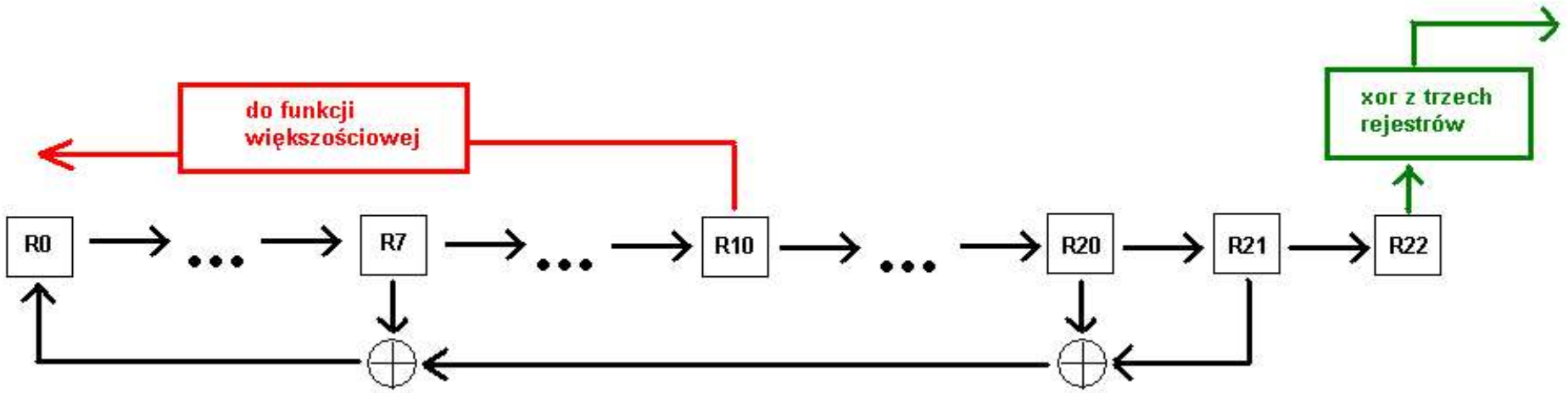
- Bezpieczeństwo,
- Jakość specyfikacji algorytmu,
- Elastyczność (od rozwiązań najefektywniejszych do najszybszych, skalowalność),
- Efektywność (zajętość, pobór mocy),
- Szybkość (przepustowość, czas inicjowania, czas re-inicjowania, częstotliwość taktowania),

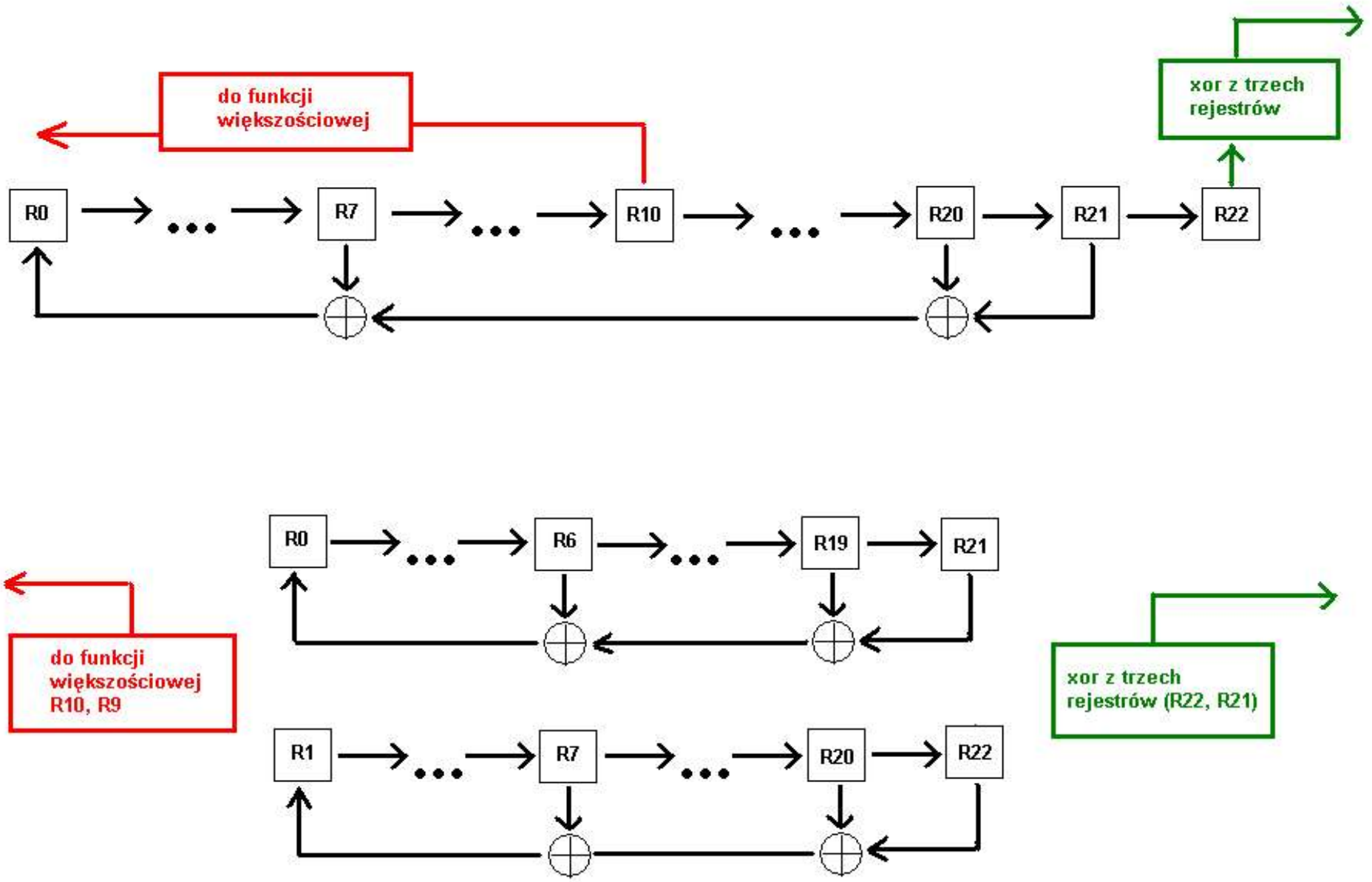
A5/1



Funkcja większościowa

Pozycje taktujące (C1, C2, C3)	*Taktowanie w następnym kroku*
(0, 0, 0)	(C1, C2, C3)
(1, 1, 1)	(C1, C2, C3)
(0, 0, 1)	(C1, C2, -)
(1, 1, 0)	(C1, C2, -)
(0, 1, 1)	(-, C2, C3)
(1, 0, 0)	(-, C2, C3)
(1, 0, 1)	(C1, -, C3)
(0, 1, 0)	(C1, -, C3)


do funkcji
większościowej
xor z trzech
rejestrów
R0
R7
R10
R20
R21
R22


do funkcji
większościowej
xor z trzech
rejestrów
R0
R7
R10
R20
R21
R22
do funkcji
większościowej
R10, R9
R0
R6
R19
R21
xor z trzech
rejestrów (R22, R21)
R1
R7
R20
R22

Funkcja większościowa – 1 runda

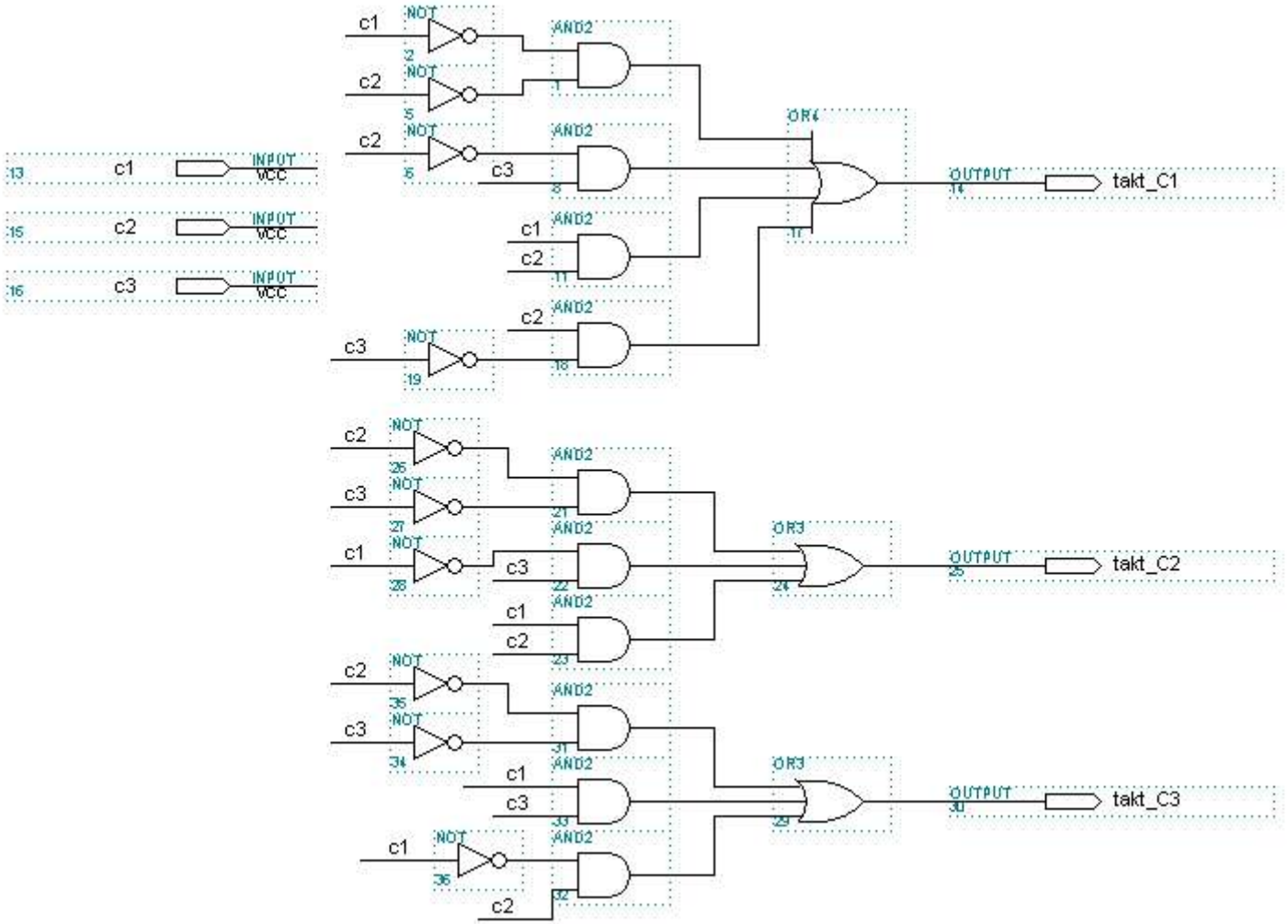
```
state[] = a1, b1, c1
switch ( state[] )
        case 000 => maj_func[] = 1 1 1
        case 111 => maj_func[] = 1 1 1

        case 001 => maj_func[] = 1 1 0
        case 110 => maj_func[] = 1 1 0

        case 011 => maj_func[] = 0 1 1
        case 100 => maj_func[] = 0 1 1

        case 101 => maj_func[] = 1 0 1
        case 010 => maj_func[] = 1 0 1
end case;
```

Funkcja większościowa – sieć bramek



Zrównoleglenie funkcji większościowej

```
state[] = (a1, b1, c1);

if ((state[] = 000) # (state[] = 111)) then
        2state[] = (a0, b0, c0);

        switch ( 2state[] )
                case 000 => takt[] = 10 10 10
                case 111 => takt[] = 10 10 10

                case 001 => takt[] = 10 10 01
                case 110 => takt[] = 10 10 01

                case 011 => takt[] = 01 10 10
                case 100 => takt[] = 01 10 10

                case 101 => takt[] = 10 01 10
                case 010 => takt[] = 10 01 10
end if;
```

Zrównoleglenie funkcji większościowej

```
state[] = (a1, b1, c1);

if ((state[] = 101) # (state[] = 010)) then
        2state[] = (a0, b1, c0);

        switch ( 2state[] )
                case 000 => takt[] = 10 01 10
                case 111 => takt[] = 10 01 10

                case 001 => takt[] = 10 01 01
                case 110 => takt[] = 10 01 01

                case 011 => takt[] = 01 01 10
                case 100 => takt[] = 01 01 10

                case 101 => takt[] = 10 00 10
                case 010 => takt[] = 10 00 10
end if;
```

A5 - podsumowanie

- 1 cykl/1bit
- Funkcja większościowa – 3 poziomy bramek,
- Max. częstotliwość 267Mhz, przepustowość – 267Mb/s,
- ok. 150 LE (Cyclone),

- 1cykl/2bity
- Funkcja wiekszościowa – 9 poziomów bramek,
- Max. czestotliwość – 145 Mhz, przepustowość – 290 Mb/s,
- ok. 500 LE,

- KRYTYCZNY ELEMENT A5/1 – NIEREGULARNE TAKTOWANIE

Ewolucja konstrukcji szyfrów strumieniowych

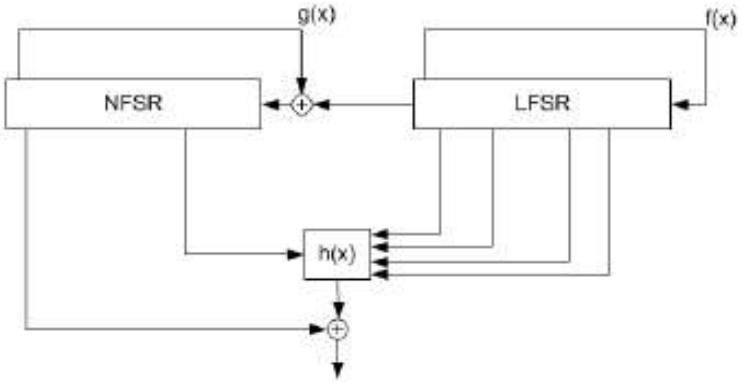
- E0, RC4, A5/1 ... standardy ?
- AES, NESSIE, CRYPTREC ... standardy ?
- Struminień klucza -> 1 bit, 8, 16, 32, 64, 128 (Rabbit) !!!
- Operacje arytemtyczne: 8, 16, 32, 64 bitowe, ... odejmowanie 1024 bitowe (HC-256) !!!

Długość słowa / stan wewnętrzny

algorytm strumieniowy	długość strumienia klucza	wielkość stanu wewnętrzngo
A5	1	64
BMGL	16	17*128 (2176)
E0	1	128+2 (130)
Helix	32	128
Leviathan	32	128
Lili-128	1	128
Mir-1	64	2*64 + 4*64 (384)
Mugi	64	3*64 + 16*64 (1216)
Rabbit	128	8*32+8*32+1 (513)
RC4	8	(8*256) 2048
Sober-16	8	128
VMPC	8	(8*256) 2048
W7	8	8*128
F8	64	128
Rijndael – OFB	128	128

Grain - eSTREAM

- Finalista eSTREAM,
- Profil sprzętowo-zorientowany,
- NFSR, LFSR – 80bitów,
- Stan wewnętrzny – 160 bitów,
- Klucz – 80 bitów,
- IV - 64 bity,



Grain – f(x), g(x), h(x)

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}$$

$$s_{i+80} = s_{i+62} + s_{i+51} + s_{i+38} + s_{i+23} + s_{i+13} + s_i.$$

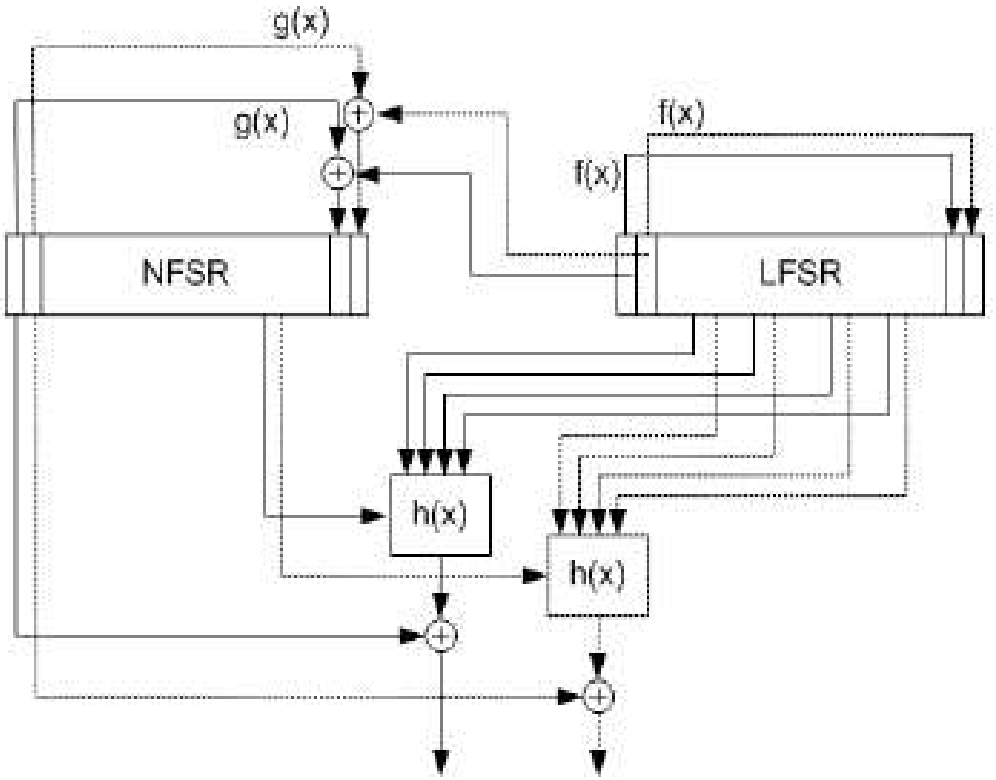
$$\begin{aligned} g(x) = 1 &+ x^{17} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{65} + x^{71} + x^{80} + \\ &+ x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + \\ &+ x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + \\ &+ x^{28}x^{35}x^{43}x^{47}x^{52}x^{59}. \end{aligned}$$

$$\begin{aligned} b_{i+80} = s_i &+ b_{i+63} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21} + \\ &+ b_{i+15} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} + \\ &+ b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9} + \\ &+ b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} + \\ &+ b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} + \\ &+ b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}. \end{aligned}$$

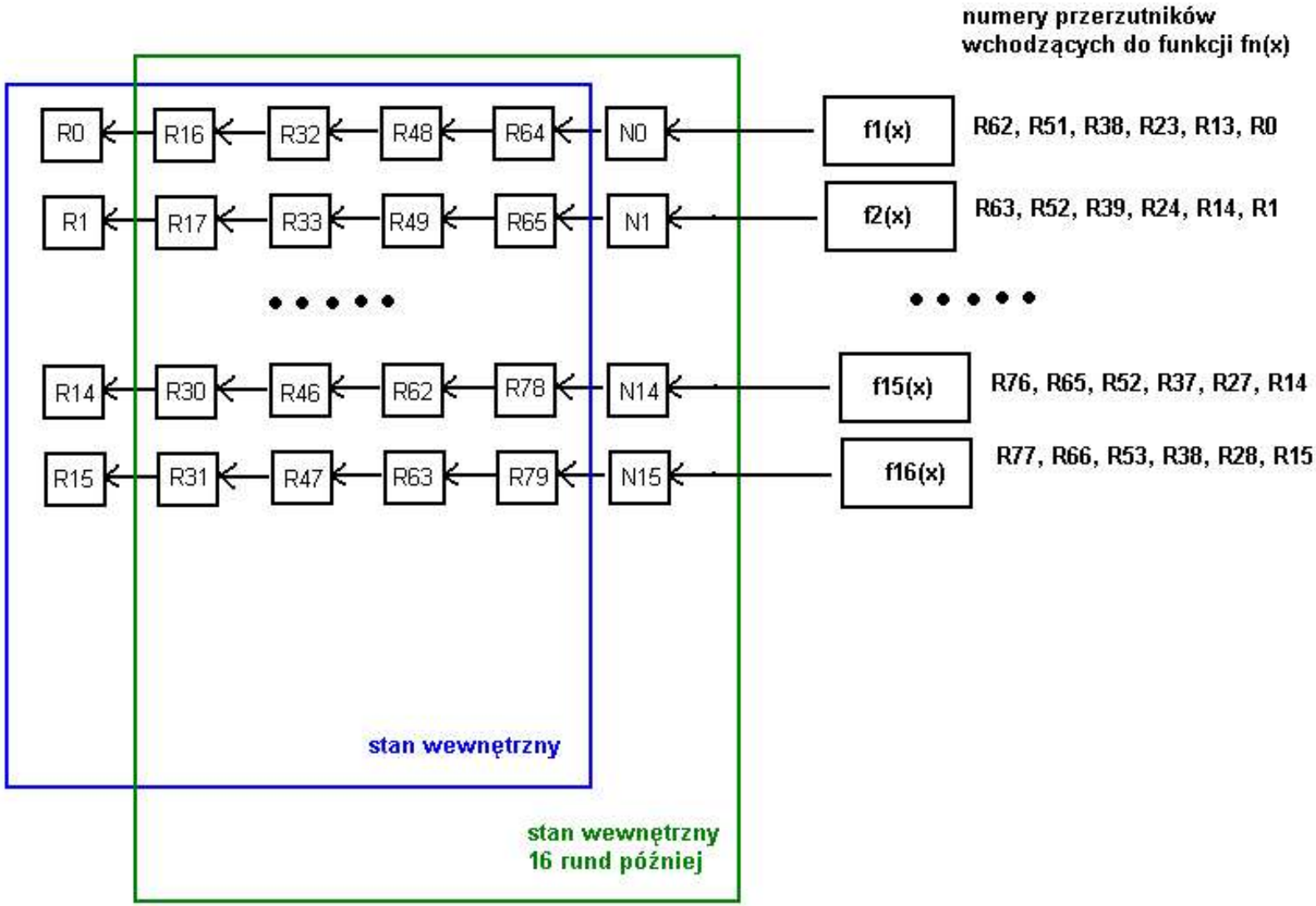
$$h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4$$

$$\begin{aligned} x_0 &\leftarrow s_{i+3} \\ x_1 &\leftarrow s_{i+25} \\ x_2 &\leftarrow s_{i+46} \\ x_3 &\leftarrow s_{i+64} \\ x_4 &\leftarrow b_{i+63} \end{aligned}$$

Grain – równolegle 2 rundy



Grain – LFSR - 16 rund w 1 cyklu



Wyniki implementacji Grain

Grain – FPGA (Cyclone)		
radix	zajętość	przepustowość
1	203 LE	155 Mb/s
2	221 LE	310 Mb/s
4	258 LE	616 Mb/s
8	334 LE	1.24 Gb/s
16	505 LE	2.48 Gb/s

Grain – ASIC		
radix	zajętość	przepustowość
16	-	4.475 Gb/s

Grain - podsumowanie

- regularne taktowanie,
- efektywność funkcji f(x), g(x), h(x),
- argumenty funkcji,
- funkcje skladowe – podstawowe operacje logiczne,

eSTREAM - Trivium

- 3 rejestry przesuwne: 93, 84, 111 bitów,
- 288 bitow stan wewnetrzny,
- Operacje skladowe: xor, and,
- Operacje skladowe: 15 in, 3 out,
- Klucz: 80 bitow, IV: 64 bity

Trivium – 1 runda

1. T1 = s66 + s93
2. T2 = s162 + s177
3. T3 = s243 + s288

4. z1 = T1 + T2 + T3 /* wyznaczenie wyjścia algorytmu*/

5. T1 = T1 + s91*s92 + s171
6. T2 = T2 + s175*s176 + s264
7. T3 = T3 + s286*s287 + s69

{s1, s2, ... s92, s93} <= {T3, s1, s2, ..., s91, s92} /*utworzenie nowego stanu*/
{s94, s95, ... s176, s177} <= {T1, s94, s95, ..., s175, s176}
{s178, s179, ... s287, s288} <= {T2, s178, s179, ..., s286, s287}

Trivium – 64 rundy w 1 cyklu

```
T1[63..0] = s[66..3] + s[93..30]
T2[63..0] = s[162..99] + s[177..114]
T3[63..0] = s[243..180] + s[288..224]
/* 64 bitowe slowo na wyjsciu algorytmu */
z1[63..0] = T1[63..0] + T2[63..0] + T3[63..0]

T1[63..0] = T1[63..0] + s[91..28]    * s[92..29]    +  s[171..108]
T2[63..0] = T2[63..0] + s[175..112] * s[176..113] +  s[264..201]
T3[63..0] = T3[63..0] + s[286..223] * s[287..224] +  s[69..6]
/* 3 x 64 bity nowych wartości*/
{s1, s2, ... s92, s93} <= {T3[0..63], s1, s2, ..., s28, s29}
{s94, s95, ... s176, s177} <= {T1[0..63], s94, s95, ..., s112, s113}
{s178, s179, ... s287, s288} <= {T2[0..63], s178, s179, ..., s223, s224}
```

Wyniki implementacji Trivium

Trivium – FPGA (Cyclone)		
radix	zajętość	przepustowość
1	324 LE	160 Mb/s
2	330 LE	314 Mb/s
4	330 LE	628 Mb/s
8	330 LE	1.24 Gb/s
16	330 LE	2.48 Gb/s
32	514 LE	4.96 Gb/s
64	706 LE	9.86 Gb/s

Trivium – ASIC		
radix	zajętość	przepustowość
64	-	18.75 Gb/s

Podsumowanie

- “Elastyczniej, efektywniej, szybciej znaczy lepiej”,
- Stan wewnętrzny < 384 bitów,
- Efektywność podstawowych funkcji,
- Podstawowe operacje: logiczne, stale rotacje, etc.

Dziękuję za uwagę ...