Белорусский государственный университет
информатики и радиоэлектроники

4-ый курс специальности ПОИТ

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЦИФРОВОГО ПРОЕКТИРОВАНИЯ

2020

Программное Обеспечение Цифрового Проектирования

Sequential Logic: shift registers :: LFSR :: advanced JC_TB

```
47    -- Clock period definitions
48    constant CLK_hperiod : time := 10 ns;
49    constant CLK_period : time := 2*CLK_hperiod;
50
51  BEGIN
52
53    -- Instantiate the Unit Under Test (UUT)
54    uut: JC_n PORT MAP (
55           CLK  => CLK,
56           RST  => RST,
57           LS   => LS,
58           Pin  => Pin,
59           Pout => Pout
60         );
61
62    -- Stimulus process
63    CLK <= not CLK after CLK_hperiod;
```

```
62    -- Stimulus process
63    CLK <= not CLK after CLK_hperiod;
64
65    stim_proc: process
66    variable test_init : std_logic_vector(0 to 3):="1111";
67    begin
68       assert (false) report "Begin of verification" severity note;
69       assert (false) report "Reset procedure" severity note;
70       wait for 2*CLK_period;
71       RST <= '1'; wait for 2*CLK_period;
72       RST <= '0'; wait for 2*CLK_period;
73       assert (Pout="0000") report "Bad reset" severity failure;
74       report "Good reset" severity note;
75
76       report "Initialization procedure" severity note;
77       Pin <= test_init; wait for CLK_period;
78       assert (LS='0' and Pout=test_init) report "Bad initialization" severity failure;
79       report "Good initialization" severity note;
80
81       report "Begin of shift procedure" severity note;
82       wait for CLK_hperiod/2;
83       LS <= '1';
84       wait for 8*CLK_period;
85       report "End of shift procedure" severity note;
86       assert (false) report "End of verification" severity failure;
87
88    end process;
```

Программное Обеспечение Цифрового Проектирования

Sequential Logic: shift registers :: LFSR :: advanced JC_TB

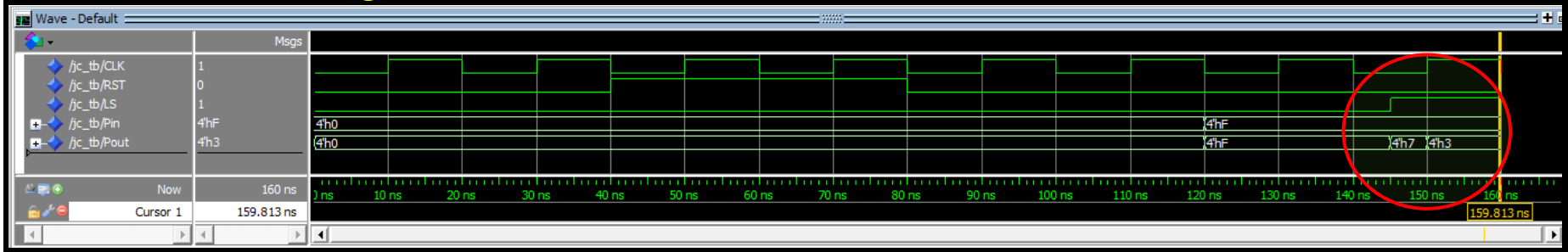
```
90    monitor_proc:process(CLK,LS,Pout)
91    variable l_val, p_val, c_val : std_logic_vector(0 to 3);
92    begin
93       if (falling_edge(CLK)) then
94          if LS='0' then
95             l_val := Pout;
96             p_val := not(l_val(3)) & l_val(0 to 2);
97          else
98             c_val := Pout;
99             assert (c_val=p_val) report "Wrong current seed" severity failure;
100            l_val := c_val;
101            p_val := not(l_val(3)) & l_val(0 to 2);
102         end if;
103      end if;
104   end process;
```

ModelSim transcript:

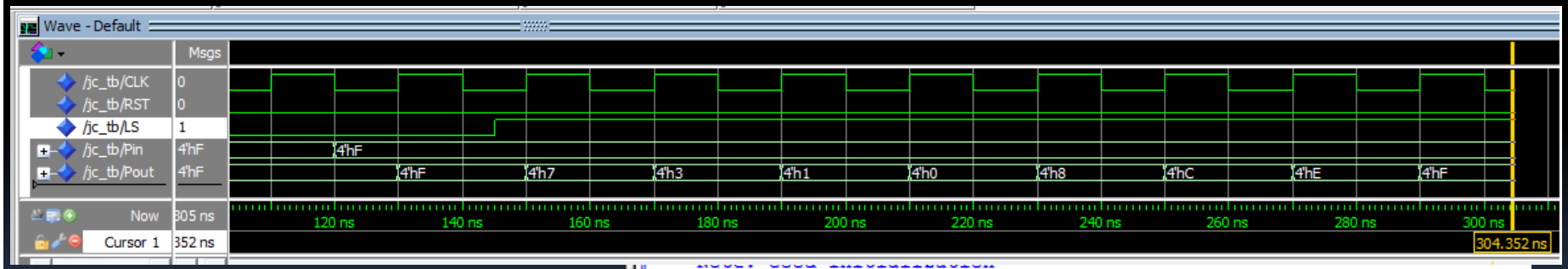
```
# ** Note: Begin of verification
#    Time: 0 ps  Iteration: 0  Instance: /jc_tb
# ** Note: Reset procedure
#    Time: 0 ps  Iteration: 0  Instance: /jc_tb
# ** Note: Good reset
#    Time: 120 ns  Iteration: 0  Instance: /jc_tb
# ** Note: Initialization procedure
#    Time: 120 ns  Iteration: 0  Instance: /jc_tb
# ** Note: Good initialization
#    Time: 140 ns  Iteration: 0  Instance: /jc_tb
# ** Note: Begin of shift procedure
#    Time: 140 ns  Iteration: 0  Instance: /jc_tb
# ** Failure: Wrong current seed
#    Time: 160 ns  Iteration: 0  Process: /jc_tb/monitor_proc File: JC_TB.vhd
# Break in Process monitor_proc at JC_TB.vhd line 99
# Simulation Breakpoint: Break in Process monitor_proc at JC_TB.vhd line 99
```

Sequential Logic: shift registers :: LFSR :: advanced JC_TB

ModelSim Wave diagram:

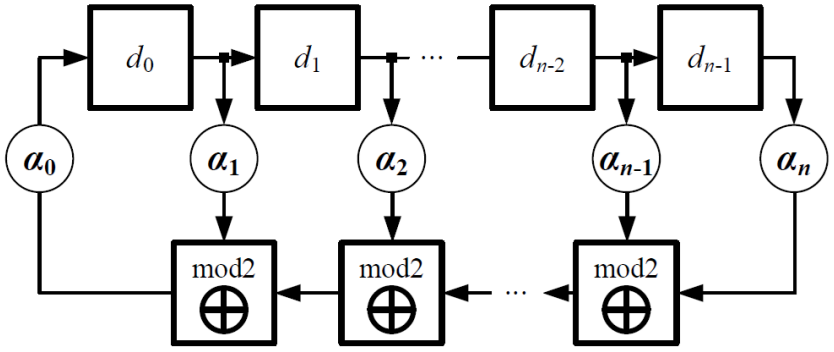



```
58    -- передача на выходной порт
59    -- символа последовательности Джонсона
60    --Pout <= sdat; -- Error!
61    Pout <= sreg;
62  End Beh;
```




```
#    Time: 140 ns  Iteration: 0  Instance: /jc_tb
# ** Note: Begin of shift procedure
#    Time: 140 ns  Iteration: 0  Instance: /jc_tb
# ** Note: End of shift procedure
#    Time: 305 ns  Iteration: 0  Instance: /jc_tb
# ** Failure: End of verification
#    Time: 305 ns  Iteration: 0  Process: /jc_tb/stim_proc File: JC_TB.vhd
# Break in Process stim_proc at JC_TB.vhd line 86
# Simulation Breakpoint: Break in Process stim_proc at JC_TB.vhd line 86
```

Sequential Logic: shift registers :: LFSR :: M-sequence generator



Общая структура генератора М-последовательности

$$\varphi(x) = \oplus \sum_{k=0}^{n} \alpha_k x^k.$$

$$\begin{vmatrix} d_0^{(k)} \\ d_1^{(k)} \\ \dots \\ d_{n-1}^{(k)} \end{vmatrix} = \begin{vmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_{k-1} & \alpha_k \\ 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 1 & 0 \end{vmatrix} \times \begin{vmatrix} d_0^{(k-1)} \\ d_1^{(k-1)} \\ \dots \\ d_{n-1}^{(k-1)} \end{vmatrix}$$

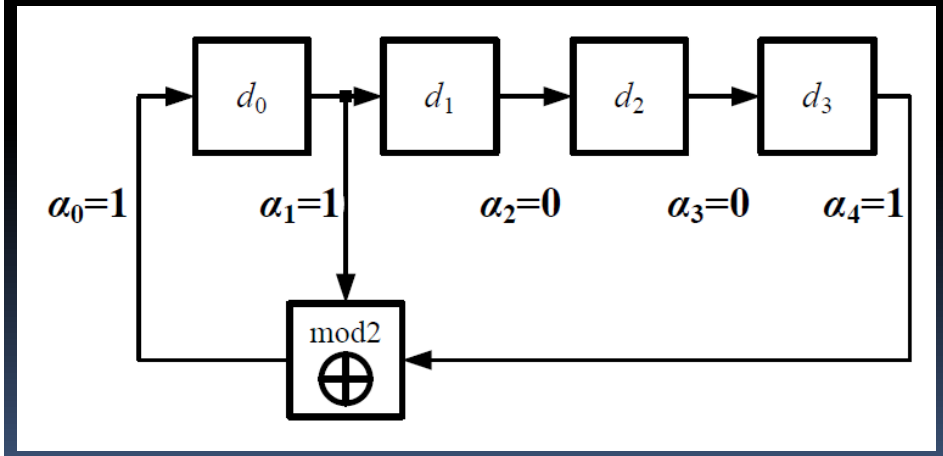
$$D^{(k)} = V D^{(k-1)}$$

$$V = \begin{vmatrix} \alpha_0 & \alpha_1 & \dots & \alpha_{k-1} & \alpha_k \\ 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \\ 0 & 0 & 0 & 1 & 0 \end{vmatrix}$$

Sequential Logic: shift registers :: LFSR :: M-sequence generator

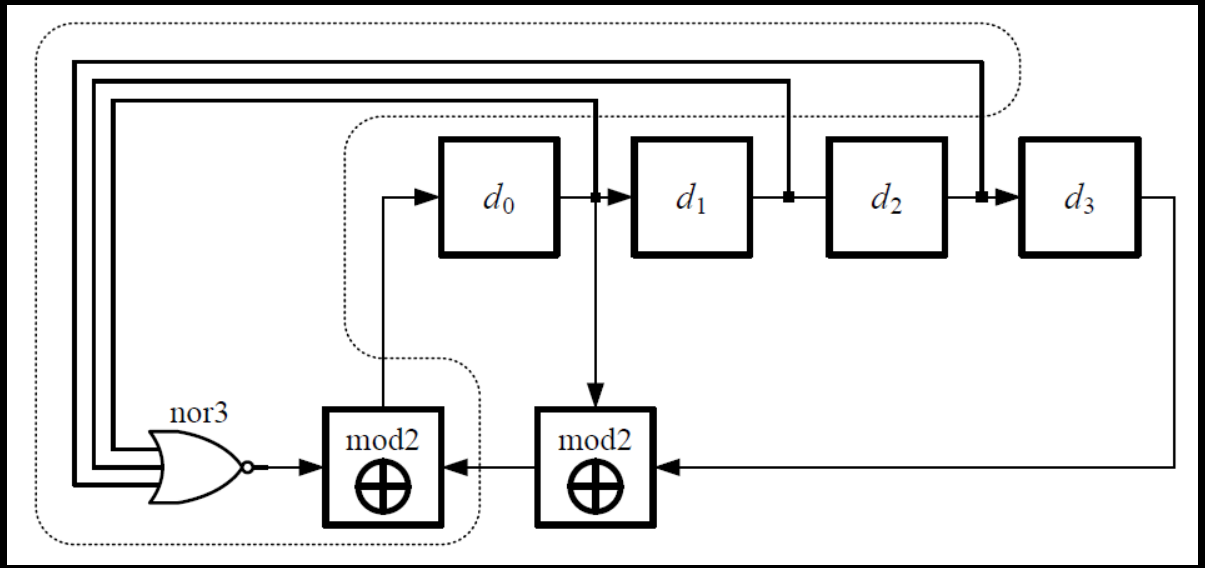
$$\varphi(x) = \oplus \sum_{k=0}^{n} \alpha_k x^k.$$

$$\varphi(x) = 1 \oplus x \oplus x^4$$



	d_0	d_1	d_2	d_3		d_0	d_1	d_2	d_3
0	1	0	0	0	8	1	1	0	1
1	1	1	0	0	9	0	1	1	0
2	1	1	1	0	10	0	0	1	1
3	1	1	1	1	11	1	0	0	1
4	0	1	1	1	12	0	1	0	0
5	1	0	1	1	13	0	0	1	0
6	0	1	0	1	14	0	0	0	1
7	1	0	1	0	15	1	0	0	0

Sequential Logic: shift registers :: LFSR :: M-sequence generator



$D^{(14)} = (0,0,0,1) \rightarrow D^{(15)} = (0,0,0,0) \rightarrow D^{(16)} = (1,0,0,0) \rightarrow D^{(17)} = (1,1,0,0) \rightarrow \ldots$

Sequential Logic: shift registers :: LFSR :: M-sequence generator

Behavioral VHDL description:

```
1  Entity LFSR_n is
2     Generic ( -- массив коэффициентов характеристического полинома,
3               -- определяет разрядность сдвигового регистра,
4               -- значение по умолчанию "11001"
5               -- соответствует полиному 4-й степени вида
6               -- f(x) = 1 + x + x^4
7               alpha : std_logic_vector := "11001" );
8      Port (   -- входной порт сигнала синхронизации
9               CLK   : in  std_logic;
10              -- входной порт сигнала инициализации
11              RST   : in  std_logic;
12              -- выходной порт символов M-последовательности
13              Pout  : out  std_logic_vector( 0 to alpha'high-1 ) );
14 End LFSR_n;
15 Architecture Beh of LFSR_n is
16 -- n-разрядный сдвиговый регистр
17 signal sreg: std_logic_vector( 0 to alpha'high-1 );
18 -- внутренняя шина данных
19 signal sdat: std_logic_vector( 0 to alpha'high-1 );
20
```

Sequential Logic: shift registers :: LFSR :: M-sequence generator

Behavioral VHDL description:

```
21  Begin
22
23    -- последовательностная схема устройства
24    Main: process( CLK, RST, sdat )
25    begin
26      -- условие асинхронной инициализации генератора
27      if RST = '1' then
28        -- начальное состояние генератора (00...0)
29        sreg <= ( others => '0' );
30      -- условие наступления переднего фронта
31      -- сигнала синхронизации
32      elsif rising_edge( CLK ) then
33        -- синхронное управление генератором
34        sreg <= sdat;
35      end if;
36    end process;
37
```

Sequential Logic: shift registers :: LFSR :: M-sequence generator

Behavioral VHDL description:

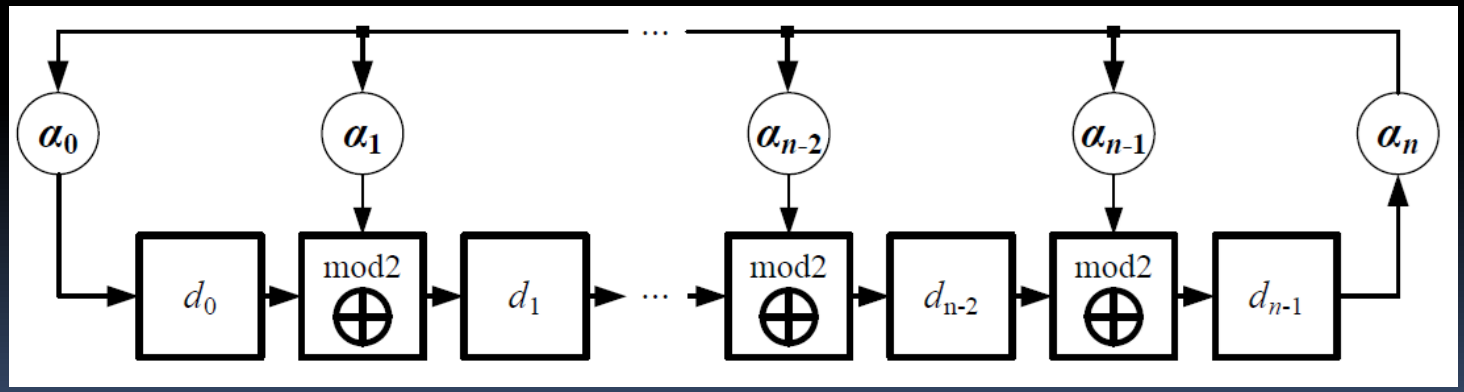
```
38    -- комбинационная схема генератора
39    Data: process( sreg )
40    -- переменная для вычисления нового значения
41    -- младшего разряда генератора
42    variable newbit : std_logic;
43    -- переменная для корректировки значения
44    -- младшего разряда с целью генерирования
45    -- нулевого состояния
46    variable zerostate: std_logic;
47    begin
48    -- инициализация переменных
49    newbit    := '0';
50    zerostate := '0';
51
52    -- вычисление значений переменных
53    for i in 0 to alpha'high-2 loop
54      zerostate := zerostate or sreg( i );
55      if alpha( i+1 ) = '1' then
56        newbit := newbit xor sreg( i );
57      end if;
58    end loop;
59    zerostate := not zerostate;
60    newbit := zerostate xor newbit xor sreg( alpha'high-1 );
61    -- формирование нового значения генератора
62    sdat <= newbit & sreg( 0 to alpha'high - 2 );
63
64    end process;
```

Sequential Logic: shift registers :: LFSR :: M-sequence generator

Behavioral VHDL description:

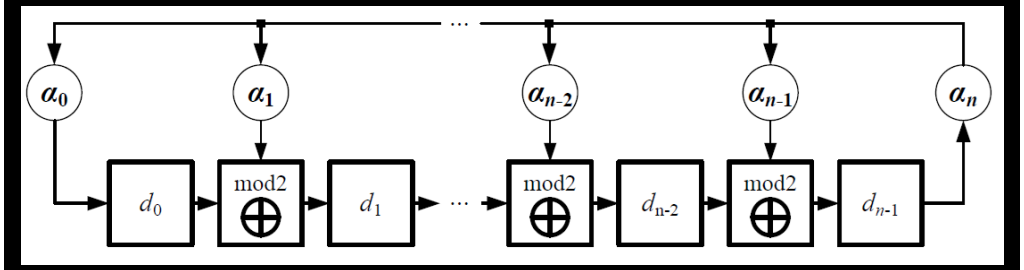
```
65
66    -- передача на выходной порт
67    -- символа М-последовательности
68  Pout <= sreg; ☺
69
70  End Beh;
```

Another structure of M-sequence generator:

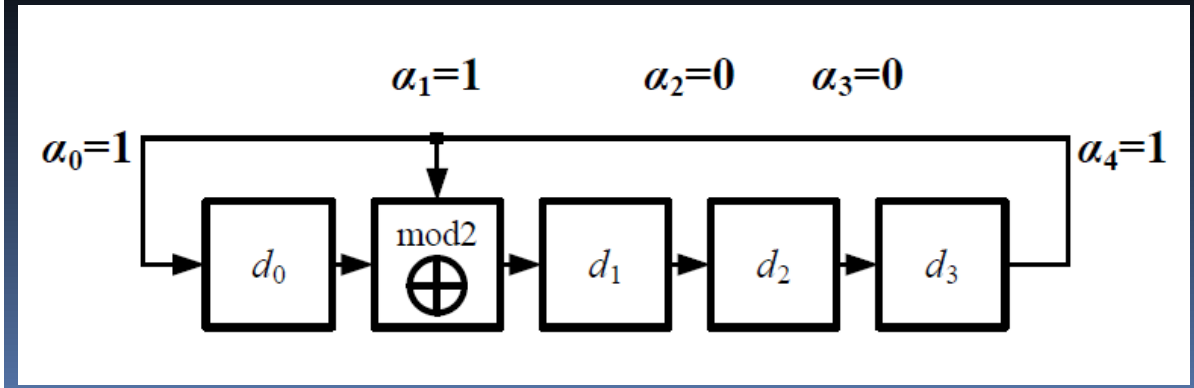


Sequential Logic: shift registers :: LFSR :: M-sequence generator

$$\varphi(x) = 1 \oplus x \oplus x^4$$



	d_0	d_1	d_2	d_3		d_0	d_1	d_2	d_3
0	1	0	0	0	8	1	0	1	0
1	0	1	0	0	9	0	1	0	1
2	0	0	1	0	10	1	1	1	0
3	0	0	0	1	11	0	1	1	1
4	1	1	0	0	12	1	1	1	1
5	0	1	1	0	13	1	0	1	1
6	0	0	1	1	14	1	0	0	1
7	1	1	0	1	15	1	0	0	0



Sequential Logic: shift registers :: LFSR :: M-sequence generator

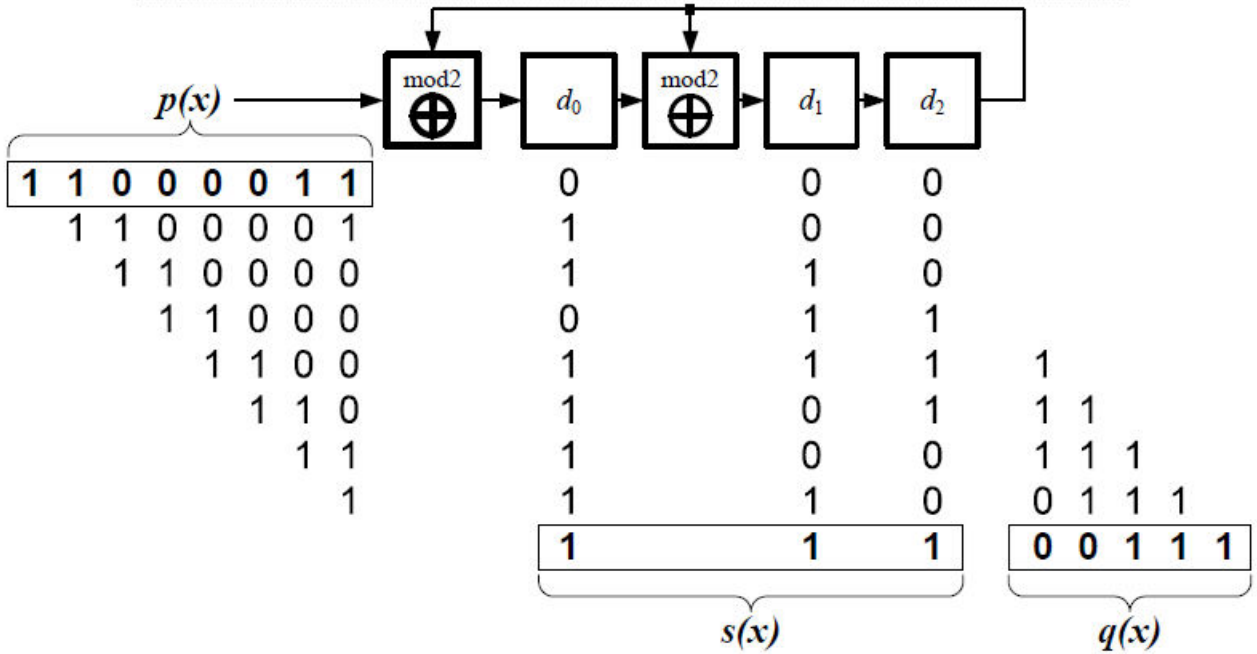
Only combinational part:

```
38    -- комбинационная схема генератора
39    Data: process( sreg )
40    begin
41    -- формирование новых значений разрядов
42    -- с учетом коэффициентов характеристического полинома
43    for i in alpha'high-1 downto 1 loop
44      if alpha( i ) = '1' then
45        sdat(i) <= sreg( alpha'high-1 ) xor sreg( i-1 );
46      else
47        sdat(i) <= sreg( i-1 );
48      end if;
49    end loop;
50    sdat( 0 ) <= sreg( alpha'high-1 );
51    end process;
52
53    -- передача на выходной порт
54    -- символа M-последовательности
55  Pout <= sreg;
56
57  End Beh;
```

Sequential Logic: shift registers :: LFSR :: Signature Analyzers

$$p(x) = q(x)\varphi(x) \oplus s(x)$$

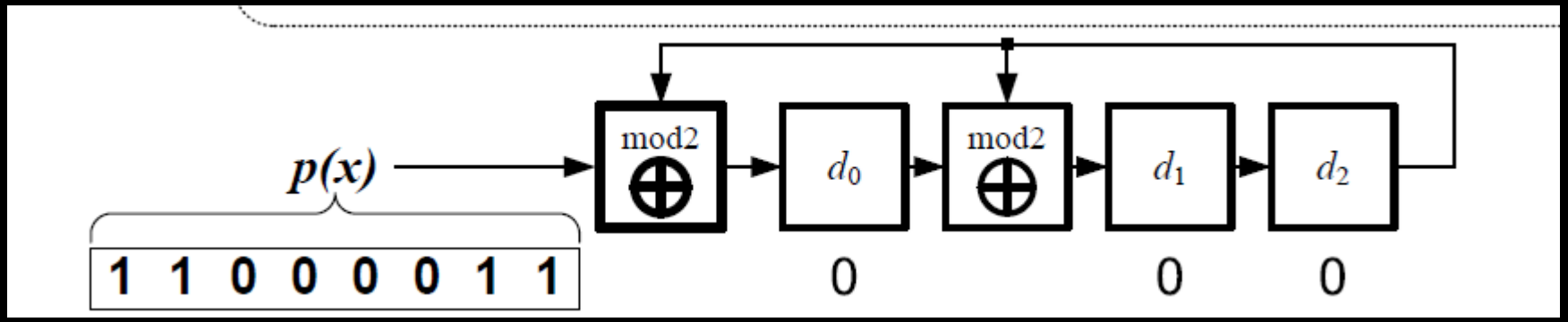
$$x^7 \oplus x^6 \oplus x \oplus 1 \;\big|\; x^3 \oplus x \oplus 1$$
$$x^7 \oplus x^5 \oplus x^4 \;\big|\; x^4 \oplus x^3 \oplus x^2$$
$$x^6 \oplus x^5 \oplus x^4 \oplus x \oplus 1$$
$$x^6 \oplus x^4 \oplus x^3 \;\big|\; x \oplus 1$$
$$x^5 \oplus x^3 \oplus x \oplus 1$$
$$x^5 \oplus x^3 \oplus x^2 \;\big|\; x \oplus 1$$
$$x^2 \oplus x \oplus 1$$



$$p(x) = 1 \oplus x \oplus x^6 \oplus x^7$$

$$\varphi(x) = 1 \oplus x \oplus x^3$$

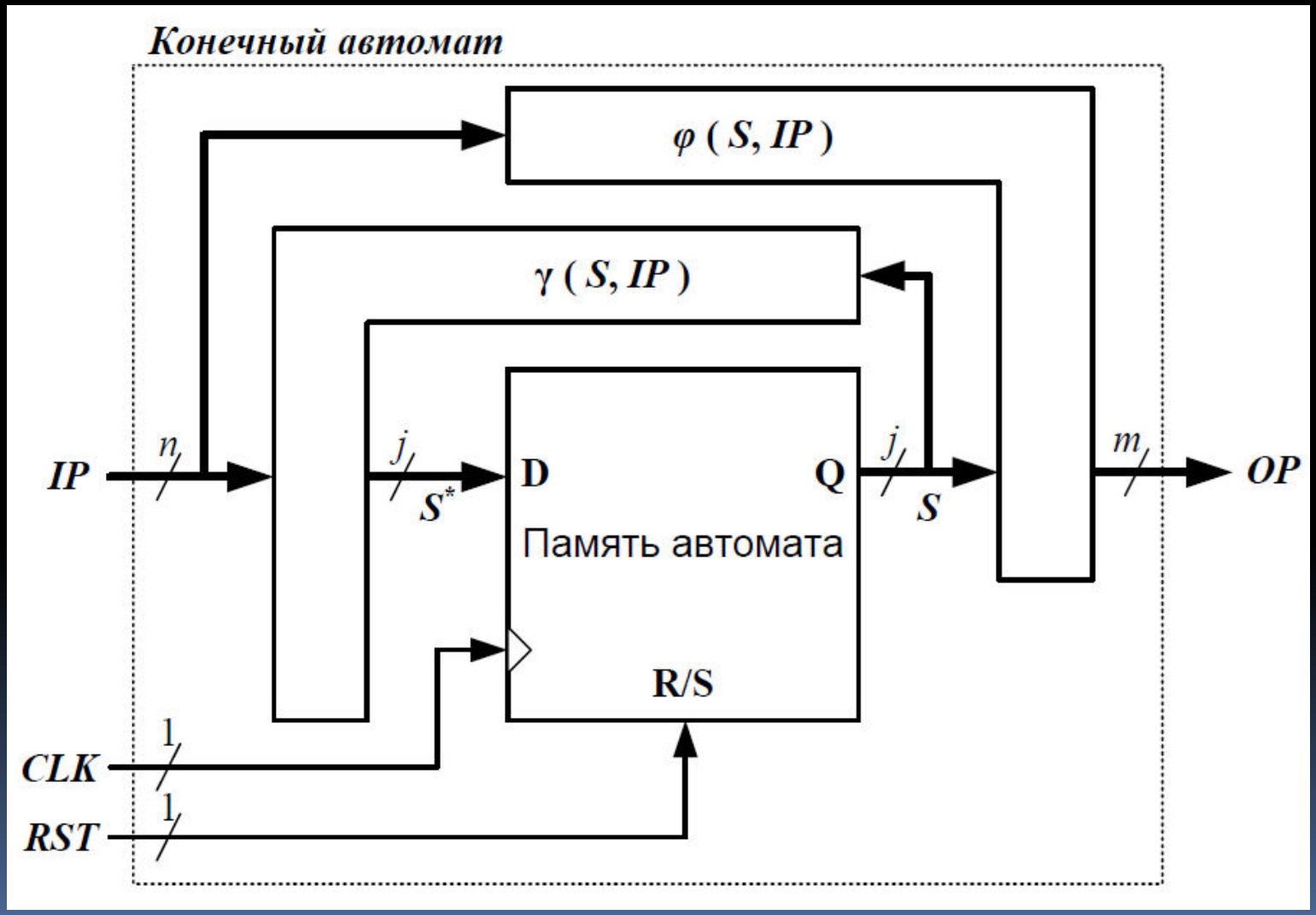
Sequential Logic: shift registers :: LFSR :: Signature Analyzers



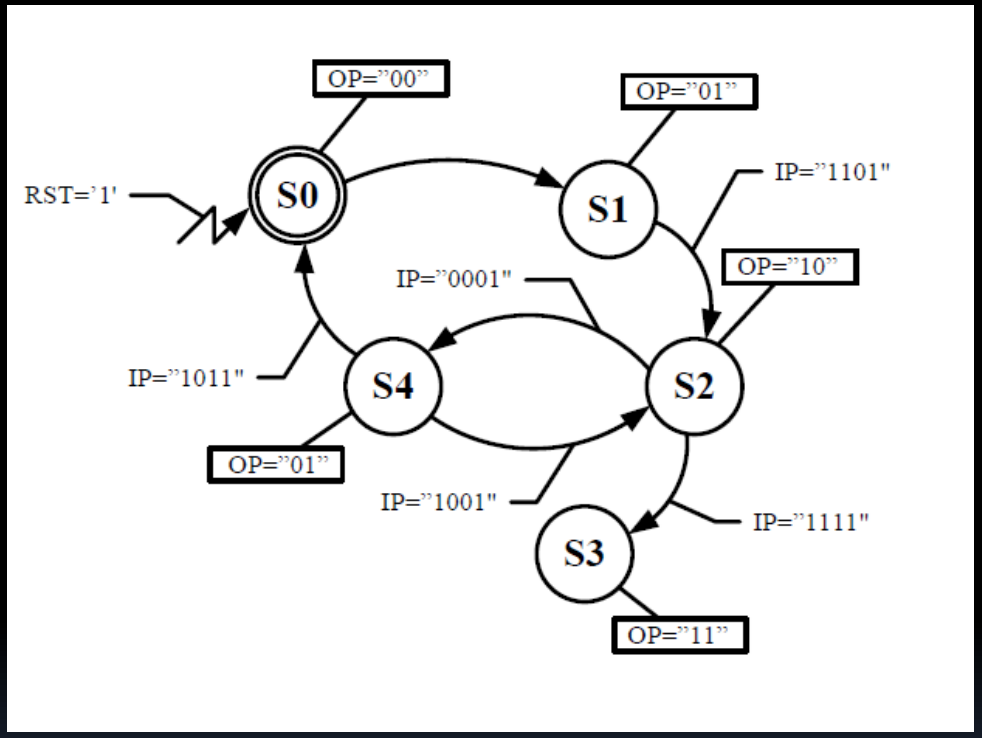

```
-- ...
        -- входной порт сжимаемых данных
        Sin : in std_logic;
-- ...
        sdat( 0 ) <= Sin xor sreg( alpha'high-1 );
-- ...
```

Программное Обеспечение Цифрового Проектирования

Sequential Logic: Finite State Machines (FSM)



Sequential Logic: Finite State Machines (FSM)



Sequential Logic: Finite State Machines (FSM)

Behavioral VHDL description:

```
1  Entity FSM is
2    Port(
3         -- Входные порты сигналов синхронизации и инициализации
4         CLK, RST : in  std_logic;
5         -- Входная шина данных
6         IP       : in  std_logic_vector( 3 downto 0 );
7         -- Выходная шина данных
8         OP       : out std_logic_vector( 1 downto 0 ) );
9  End FSM;
10
11 Architecture Beh of FSM is
12 -- перечисляемый тип состояний автомата
13 type states is ( S0, S1, S2, S3, S4 );
14 -- сигналы текущего и следующего состояния
15 signal current_state, next_state : states;
16 -- сигналы для выходных буферов-усилителей
17 signal fop : std_logic_vector( 1 downto 0 );
18
```

Sequential Logic: Finite State Machines (FSM)

Behavioral VHDL description:

```
11  Architecture Beh of FSM is
12  -- перечисляемый тип состояний автомата
13  type states is ( S0, S1, S2, S3, S4 );
14  -- сигналы текущего и следующего состояния
15  signal current_state, next_state : states;
16  -- сигналы для выходных буферов-усилителей
17  signal fop : std_logic_vector( 1 downto 0 );
18
19  Begin
20    -- Описание синхронной памяти автомата
21    FSM_dff: process( CLK, RST, next_state )
22    begin
23      -- условие инициализации автомата
24      if RST = '1' then
25        -- стартовое состояние автомата
26        current_state <= S0;
27      -- условие изменения текущего состояния
28      -- на новое
29      elsif rising_edge( CLK ) then
30        current_state <= next_state;
31      end if;
32    end process;
33
```

Sequential Logic: Finite State Machines (FSM)

Behavioral VHDL description:

```
34    -- Описание комбинационной логики,
35    -- вырабатывающей значения нового состояния
36    FSM_gamma: process( current_state, IP )
37    begin
38      -- анализ текущего состояния
39      case current_state is
40        when S0 => next_state <= S1;
41        when S1 => if IP = "1101" then
42                      next_state <= S2;
43                   else
44                      next_state <= S1;
45                   end if;
46        when S2 => if IP = "0001" then
47                      next_state <= S4;
48                   elsif IP = "1111" then
49                      next_state <= S3;
50                   else
51                      next_state <= S2;
52                   end if;
53        when S3 => next_state <= S3;
54        when S4 => if IP = "1011" then
55                      next_state <= S0;
56                   else
57                      next_state <= S4;
58                end if;
59        when others => next_state <= S0;
60      end case;
61    end process;
62
```

Sequential Logic: Finite State Machines (FSM)

Behavioral VHDL description:

```
63    -- Описание комбинационной логики,
64    -- вырабатывающей значения для выходной
65    -- шины данных
66    FSM_phi: process( current_state )
67    begin
68      case current_state is
69        when S0        => fop <= "00";
70        when S1 | S4   => fop <= "01";
71        when S2        => fop <= "10";
72        when S3        => fop <= "11";
73        when others    => fop <= "00";
74      end case;
75    end process;
76
77    -- Передача сформированных значений
78    -- на выходную шину данных
79    OP <= fop;
80  End Beh;
```