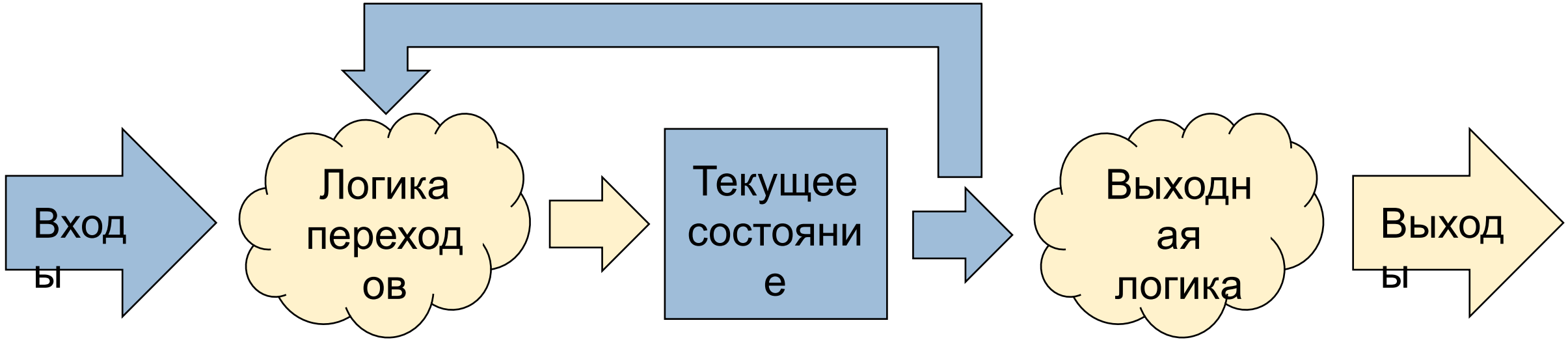
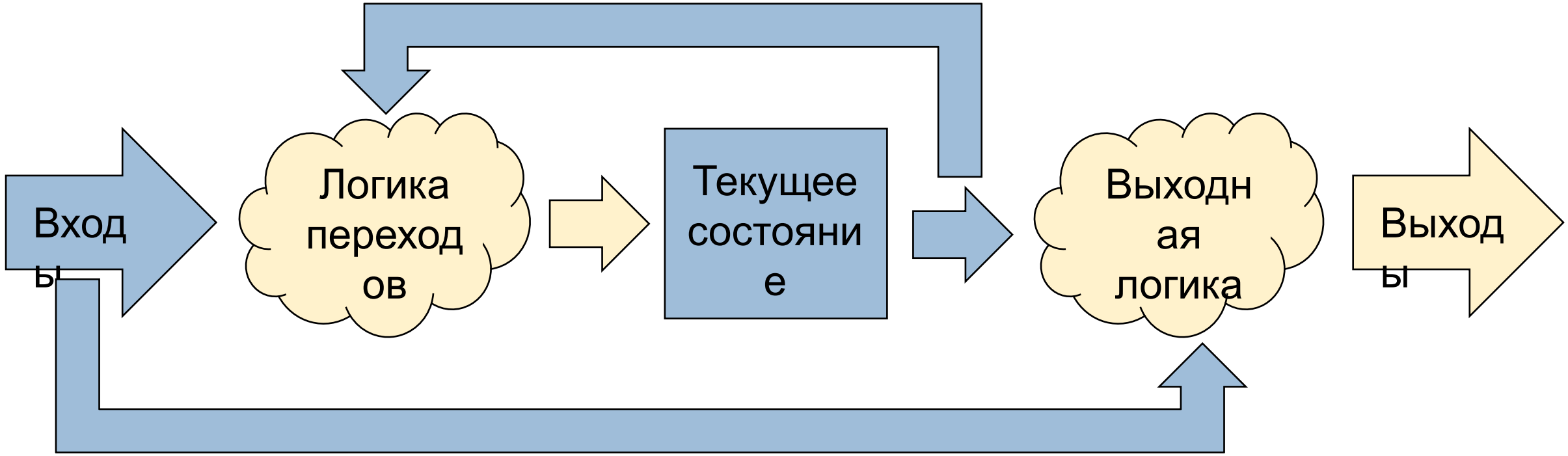
Конечные автоматы

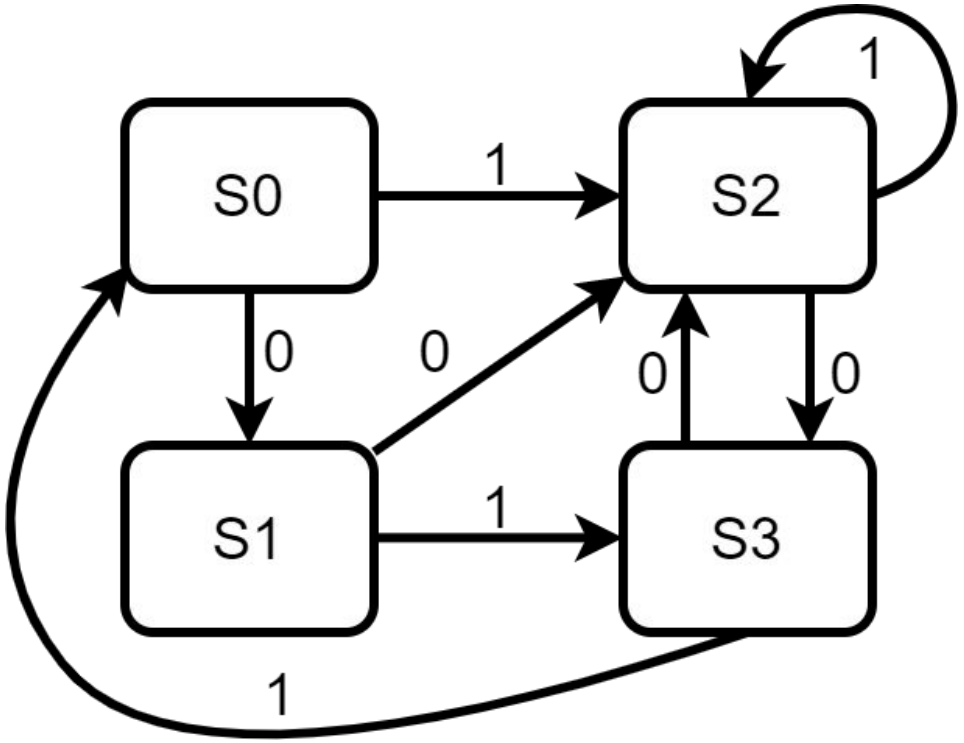
Конечные автоматы. Автомат Мура

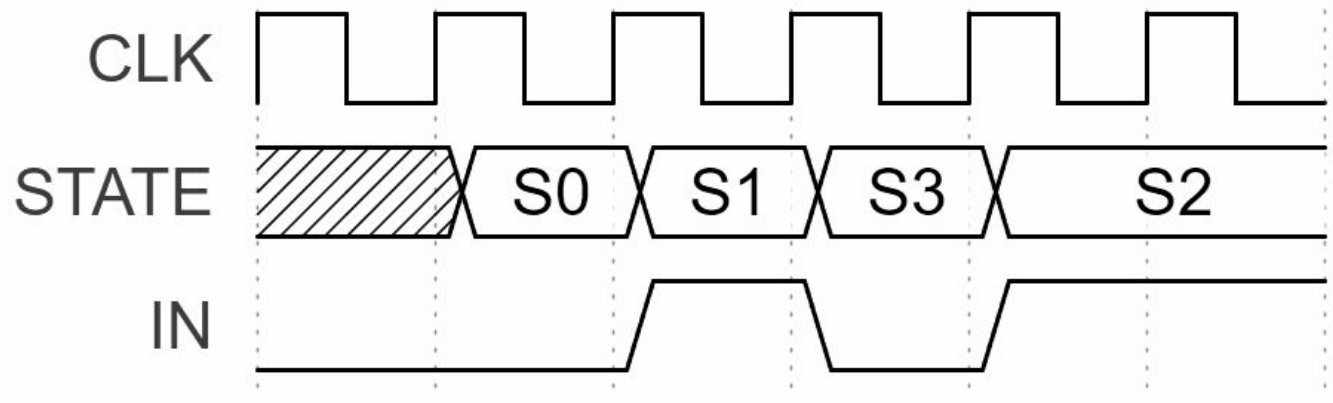


Конечные автоматы. Автомат Мили

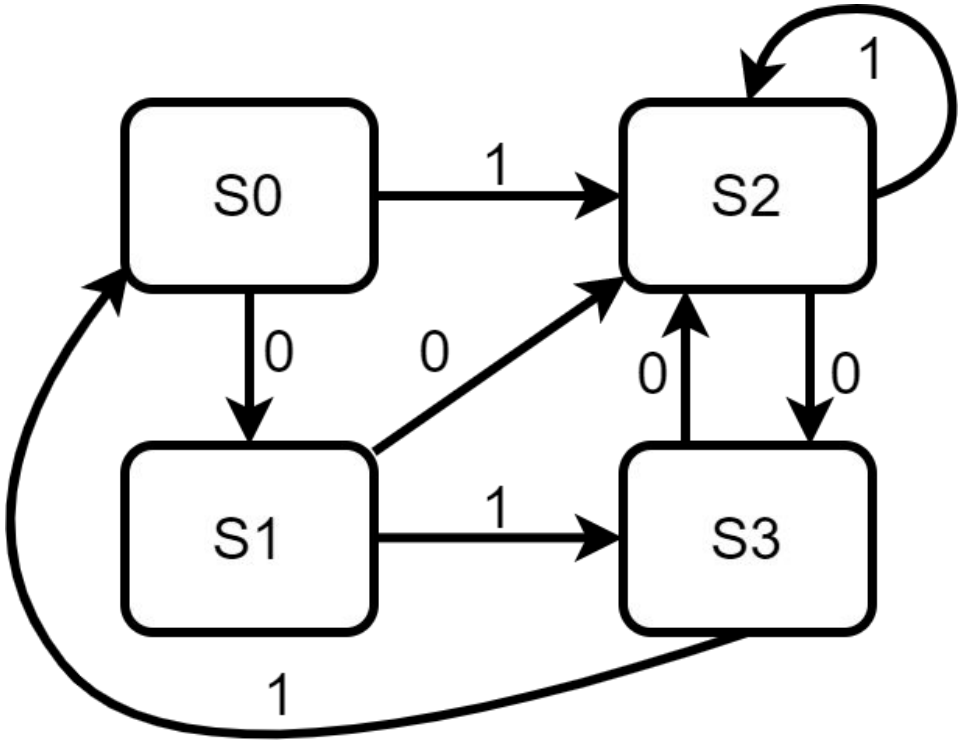


Конечные автоматы





Конечные автоматы. Verilog HDL



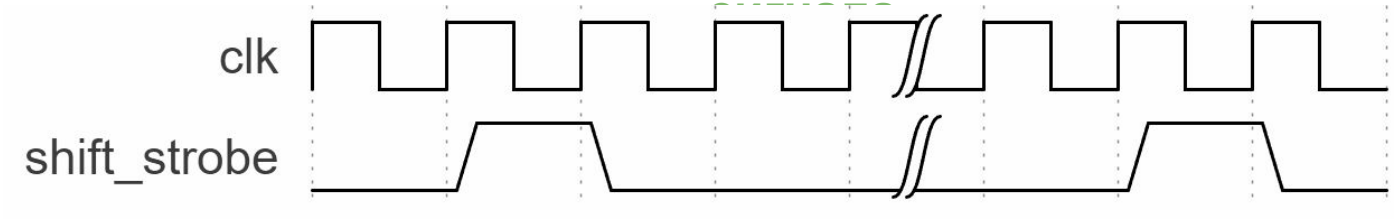

```
module my_fsm
(
    …
);
reg [1:0] STATE;
always @(posedge CLK) begin
      if (RST)
         STATE <= 2’d0;
      else begin
         case (STATE)
            2'd0: if (IN == 1'b1) STATE <= 2'd2;
                     else STATE <= 2'd1;
            2'd1: if (IN == 1'b1) STATE <= 2'd3;
                     else STATE <= 2’d2;
            ...
         endcase
      end
end
endmodule
```

Упражнение: кодовый замок

```
wire [3:0] key_db;

sync_and_debounce # (.w (4), .depth
(debounce_depth))
    i_sync_and_debounce_key
        (clk, reset, ~ key_sw, key_db);
...

wire shift_strobe;

strobe_gen # (.w (shift_strobe_width)) i_shift_strobe
   (clk, reset, shift_strobe);
```

Модуль для устранения дребезга кнопок

Модуль для генерации стробирующего



Упражнение: кодовый замок

```
wire [3:0] out_reg;

shift_register # (.w (4)) i_shift_reg
(
    .clk ( clk ),
    .reset ( reset ),
    .en ( shift_strobe ),
    .in ( key_db [3] ),
    .out_reg ( out_reg )
);

assign led = ~ out_reg;
```

4-х битный сдвиговый регистр

Если поднят «редкий» стробирующий сигнал, в регистр вдвигается значение: нажата кнопка или нет.

Значение регистра выведено на светодиоды

Упражнение: кодовый замок

```
wire [7:0] shift_strobe_count;

counter # (8) i_shift_strobe_counter
(
    .clk ( clk ),
    .reset ( reset ),
    .en ( shift_strobe ),
    .cnt ( shift_strobe_count )
);
```

8-х битный счетчик числа стробов

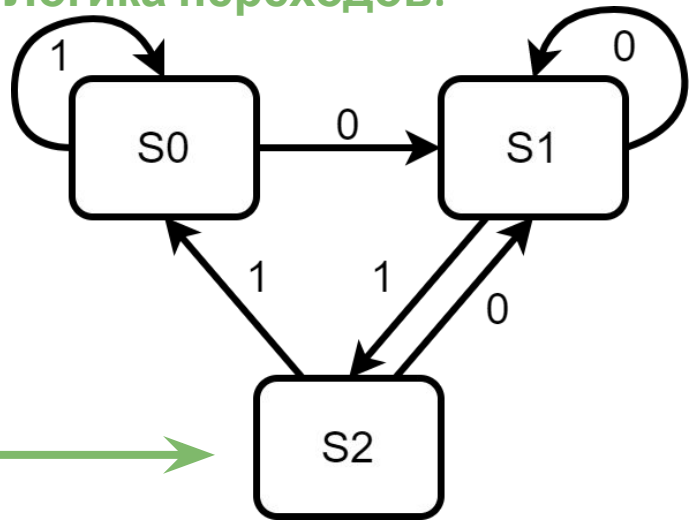
Счетчик увеличивается каждый раз, когда значение стробирующего сигнала равно единице.

Упражнение: кодовый замок

```
wire out_moore_fsm;
moore_fsm i_moore_fsm (
    .clk ( clk ),
    .reset ( reset ),
    .en ( shift_strobe ),
    .a ( out_reg [0] ),
    .y ( out_moore_fsm )
);
```

Конечный автомат Мура

Логика переходов:



Если конечный автомат в состоянии “S2”, то выход out_moore_fsm **равен “1”.**

Как долго конечный автомат может находиться в состоянии S2?

Упражнение: кодовый замок

```
wire out_moore_fsm;
moore_fsm i_moore_fsm (
    .clk ( clk ),
    .reset ( reset ),
    .en ( shift_strobe ),
    .a ( out_reg [0] ),
    .y ( out_moore_fsm )
);
```

Конечный автомат Мура

```
wire [3:0] moore_fsm_out_count;
counter # (4) i_moore_fsm_out_counter (
    .clk ( clk ),
    .reset ( reset ),
    .en ( shift_strobe & out_moore_fsm ),
    .cnt ( moore_fsm_out_count )    );
```

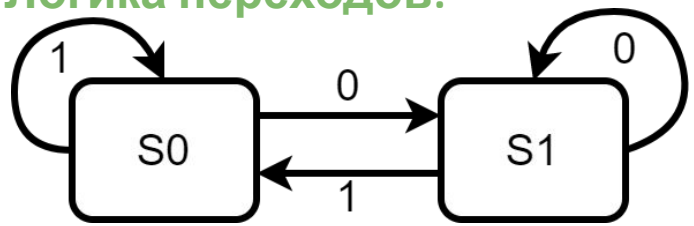
Счетчик, который срабатывает по стробирующему сигналу и выходу конечного автомата Мура

Упражнение: кодовый замок

```
wire out_mealy_fsm;
mealy_fsm i_mealy_fsm (
    .clk ( clk ),
    .reset ( reset ),
    .en ( shift_strobe ),
    .a ( out_reg [0] ),
    .y ( out_mealy_fsm )
);
```

Конечный автомат Мили

Логика переходов:



Если конечный автомат в состоянии “S1” И если входное значение равно “1”, то выход out_mealy_fsm **равен “1”.**

Упражнение: кодовый замок

```
wire out_mealy_fsm;
mealy_fsm i_mealy_fsm (
    .clk ( clk ),
    .reset ( reset ),
    .en ( shift_strobe ),
    .a ( out_reg [0] ),
    .y ( out_mealy_fsm )
);
```

Конечный автомат Мили

```
wire [3:0] mealy_fsm_out_count;
counter # (4) i_mealy_fsm_out_counter (
    .clk ( clk ),
    .reset ( reset ),
    .en ( shift_strobe & out_mealy_fsm ),
    .cnt ( mealy_fsm_out_count )
);
```

Счетчик, который срабатывает по стробирующему сигналу и выходу конечного автомата Мили

Упражнение: кодовый замок

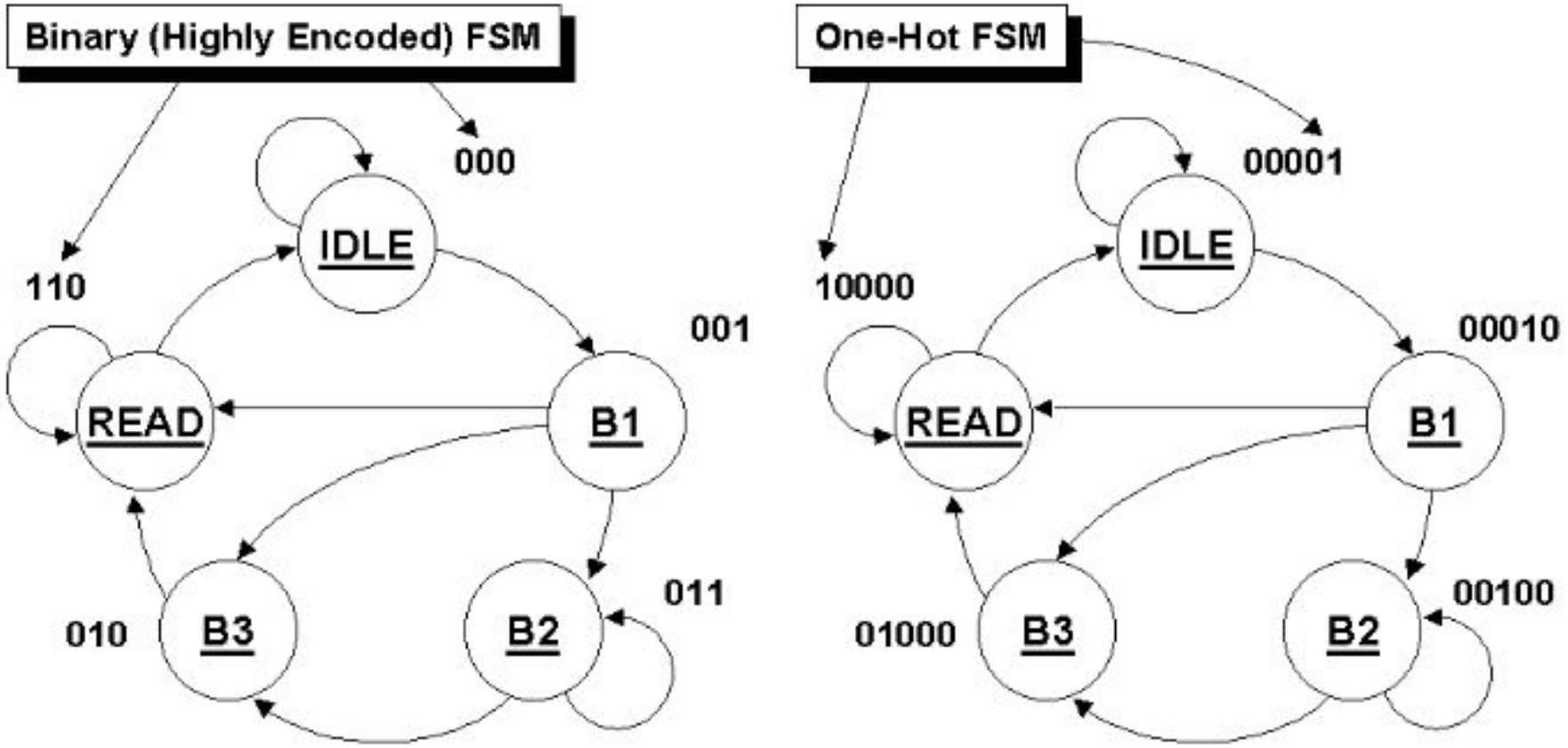
1. Модифицируйте конечный автомат Мура так, чтобы он распознавал последовательность входного сигнала **«1, 0, 1, 1, 0».**
2. Модифицируйте конечный автомат Мили так, чтобы он распознавал последовательность входного сигнала **«1, 0, 1, 1, 0».**

Конечные автоматы. Источники информации

1. Материалы для это части презентации взяты из материалов:
    1. *Clifford E. Cummings The Fundamentals of Efficient Synthesizable Finite State Machine Design using NC-Verilog and BuildGates*
    2. *Clifford E. Cummings Finite State Machine (FSM) Design & Synthesis using SystemVerilog - Part I*

Примеры кода из статьи приводятся на **SystemVerilog!** Обратите на это внимание! Тоже самое применимо и для Verilog кроме enum.

Конечные автоматы. Кодирование



Clifford E. Cummings Finite State Machine (FSM) Design & Synthesis using SystemVerilog - Part I

Конечные автоматы. Требования к описанию FSM

1. Описание конечного автомата должно быть легко модифицируемым.
2. Описание конечного автомата должно быть компактным.
3. Описание конечного автомата должно быть легким для понимания.
4. Описание конечного автомата должно облегчать отладку.
5. Описание конечного автомата должно давать эффективные результаты синтеза.

Больше кода = больше ошибок

Clifford E. Cummings Finite State Machine (FSM) Design & Synthesis using SystemVerilog - Part I

Конечные автоматы. One Always Block FSM

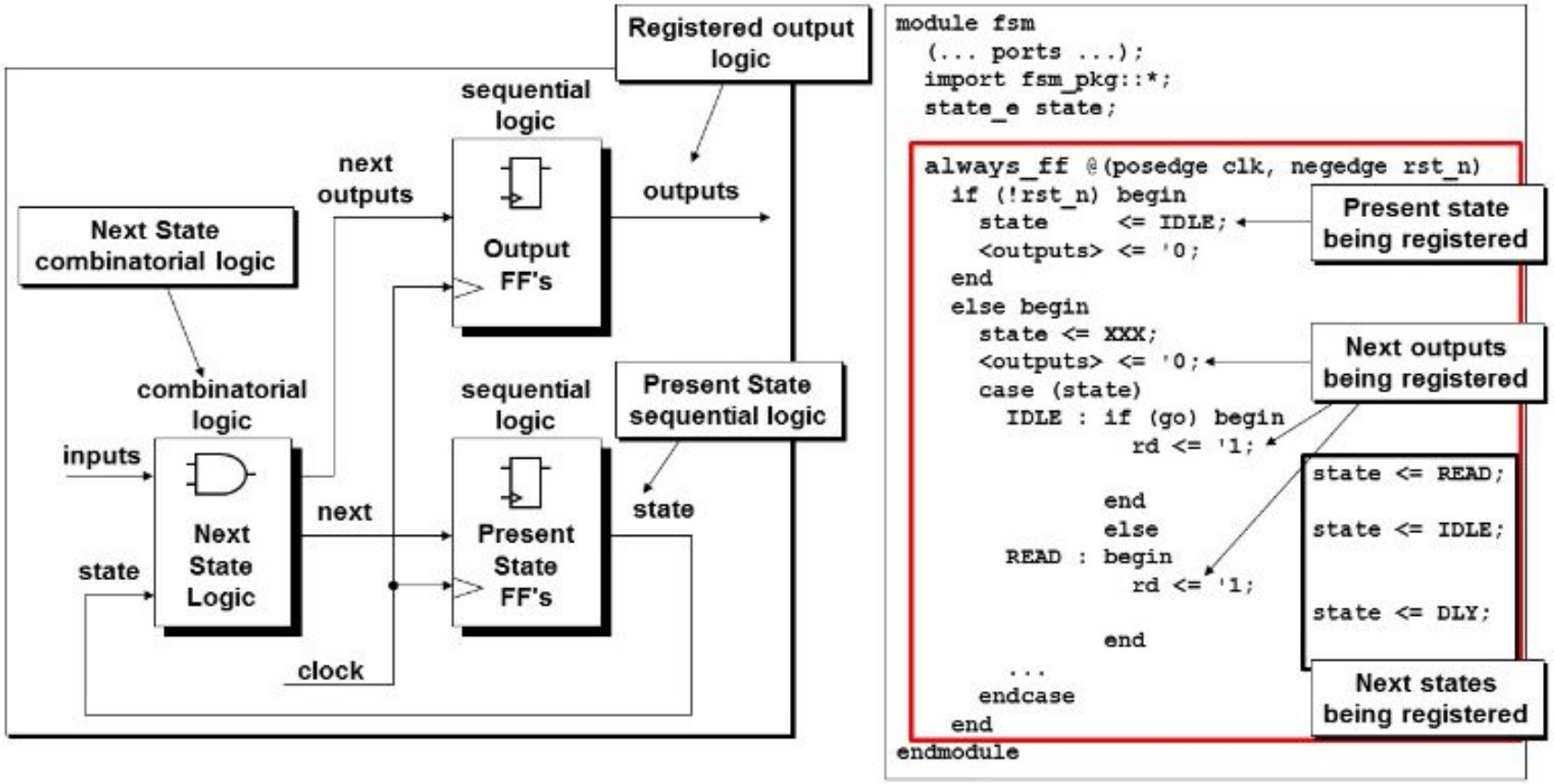
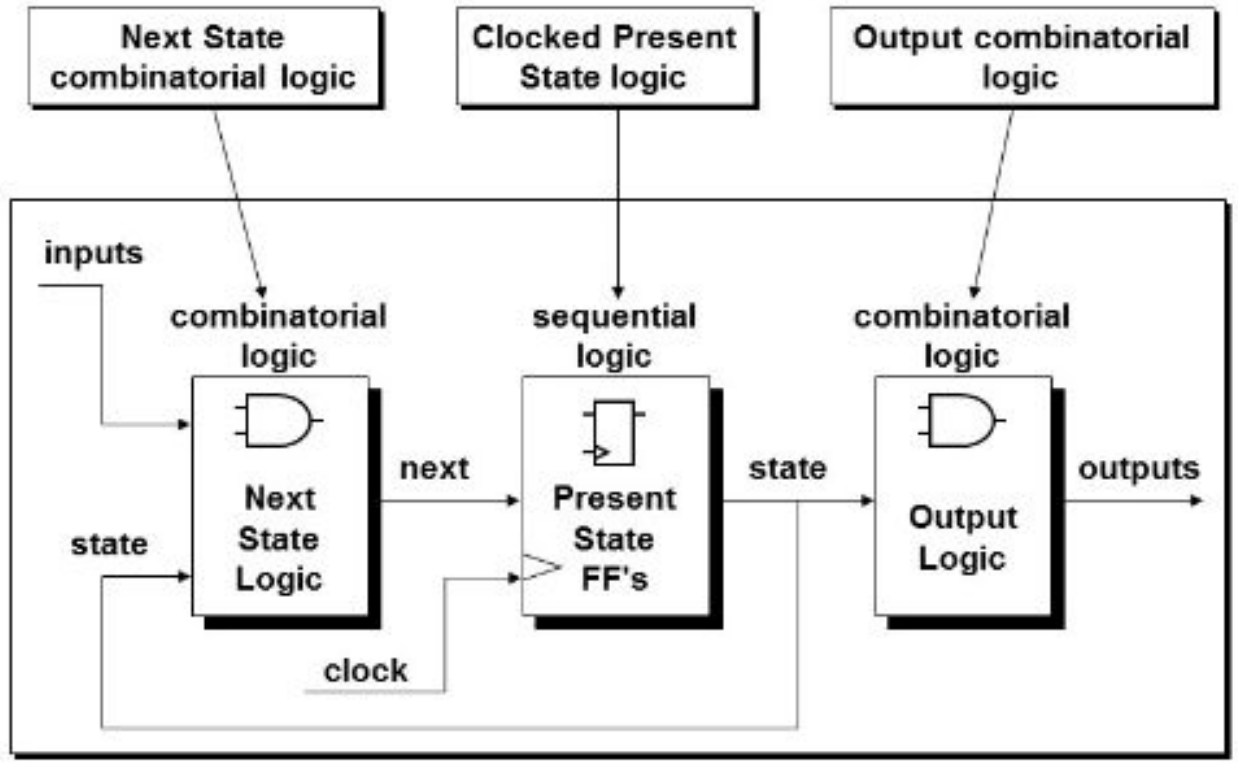


Figure 3 - Block diagram for 1-always block coding style

Clifford E. Cummings Finite State Machine (FSM) Design & Synthesis using SystemVerilog - Part I

Конечные автоматы. Two Always Block FSM



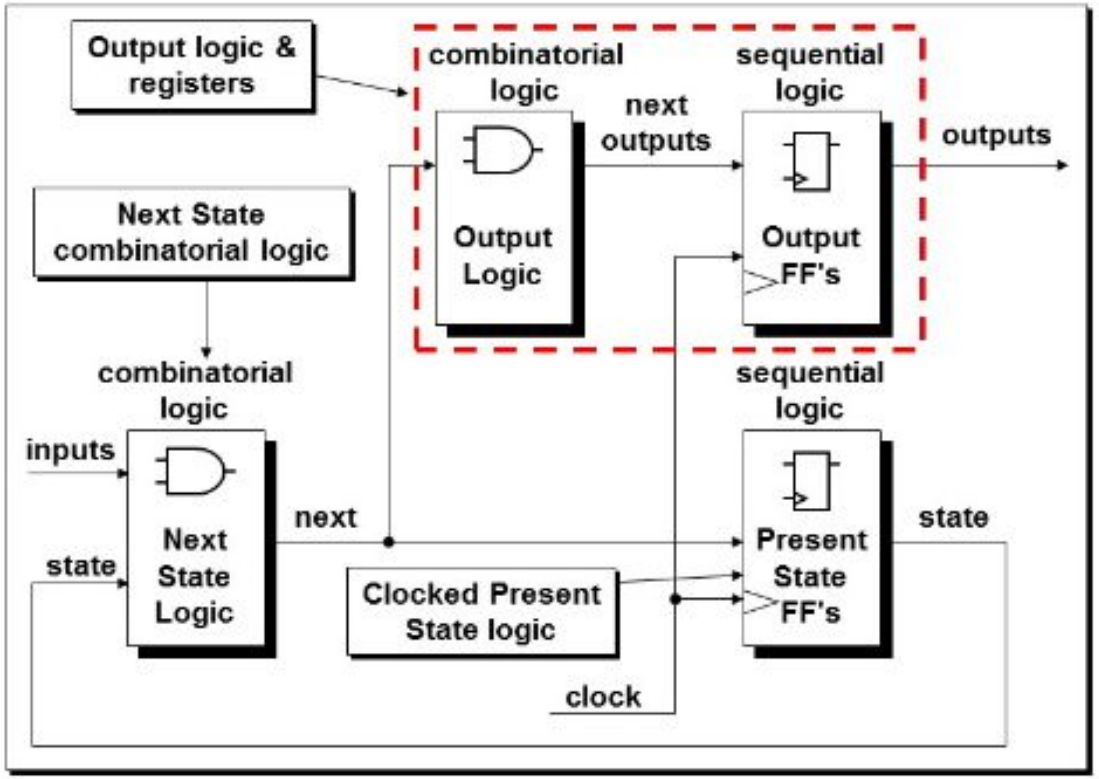

```
module fsm
  (... ports ...);
  import fsm_pkg::*;
  state_e state, next;

always_ff @(posedge clk or negedge rst_n)
  if (!rst_n) state <= IDLE;
  else        state <= next;

always_comb begin
  next = XXX;
  rd   = '0;
  ds   = '0;
  case (state)
    IDLE : if (go)      next = READ;
           else         next = IDLE;
    READ : begin
             rd = '1;
                        next = DLY;
    ...
  endcase
end
endmodule
```

Clocked present state logic

Next & output combinatorial logic

Clifford E. Cummings Finite State Machine (FSM) Design & Synthesis using SystemVerilog - Part I

Конечные автоматы. Three Always Block FSM coding style




```
module fsm
  (... ports ...);
  import fsm_pkg::*;
  state_e state, next;

  always_ff @(posedge clk or negedge rst_n)
    if (!rst_n) state <= IDLE;
    else        state <= next;

  always_comb begin
    next = XXX;
    case (state)
      IDLE : if (go)   next = READ;
             else      next = IDLE;
      READ :           next = DLY;
      ...
    endcase
  end

  always_ff @(posedge c
    if (!rst_n) begin
      rd <= '0;
      ds <= '0;
    end
    else begin
      rd <= '0;
      ds <= '0;
      case (next)
        READ : rd <= '1;
        ...
      endcase
    end
endmodule
```

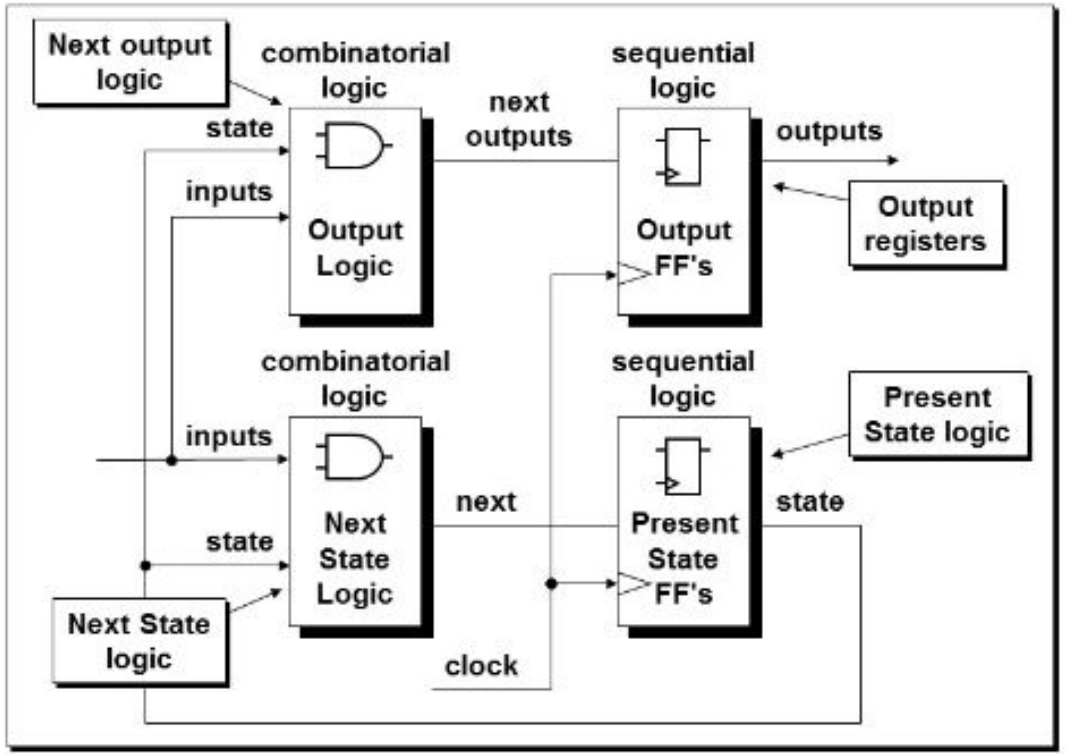
Clocked present state logic

always_comb next logic

always_ff *next*-output logic & registers

When next is READ, output will be rd<=1

Конечные автоматы. Four Always Block FSM




```
module fsm
  (... ports ...);
  import fsm_pkg::*;
  state_e state, next;

always_ff @(posedge clk or negedge rst_n)
  if (!rst_n) state <= IDLE;
  else        state <= next;

always_comb begin
  next = XXX;
  case (state)
    IDLE : if (go)  next = READ;
           else     next = IDLE;
    ...
  endcase
end

always_comb begin
    n_rd = '0;
    n_ds = '0;
  case (state)
     READ : n_rd = '1; // READ
     ...
  endcase
end

always_ff @(posedge clk or negedge rst_n)
  if (!rst_n) begin ...
  else begin
    rd <= n_rd;
    ds <= n_ds;
  end
endmodule
```

Clocked present state logic

always_comb next logic

always_comb *next*-output logic

next-outputs

always_ff output registers

Clifford E. Cummings Finite State Machine (FSM) Design & Synthesis using SystemVerilog - Part I

Конечные автоматы. Enum. Только для SystemVerilog!

```
package fsm1_pkg;
  typedef enum {IDLE,
                READ,
                DLY,
                DONE,
                XXX } state_e;
endpackage
```

```
package fsm1_pkg;
  typedef enum logic [1:0] {IDLE = 2'b00,
                            READ = 2'b01,
                            DLY  = 2'b11,
                            DONE = 2'b10,
                            XXX  = 'x   } state_e;
endpackage
```

Конечные автоматы. Регистр состояния

```
always_ff @(posedge clk, negedge rst_n)
  if (!rst_n) state <= IDLE;
  else        state <= next;
```

```
always_ff @(posedge clk, negedge rst_n) begin
  testbad <= go;
  if (!rst_n) state <= IDLE;
  else        state <= next;
end
```

```
1: Error:
<filename><line number>:
The statements in this 'always' block are outside the scope of the
synthesis policy. Only an 'if' statement is allowed at the top level in
this always block. (ELAB-302)

2: Error: Cant' read 'sverilog' file
<filename>
(UID-59)
```

Конечные автоматы. Читаемость

```
always_comb begin
  next = XXX;
  case (state)
    IDLE : if (go)   next = READ;
           else      next = IDLE; //@ loopback
    READ :           next = DLY;
    DLY  : if (!ws)  next = DONE;
           else      next = READ;
    DONE :           next = IDLE;
    default:         next = XXX;
  endcase
end
```

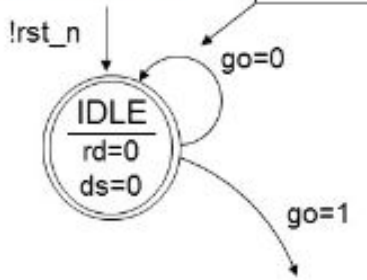
```
always_comb begin
  next = XXX;
  case (state)
    IDLE : if (go) next = READ;
           else next = IDLE; //@ loopback
    READ : next = DLY;
    DLY  : if (!ws) next = DONE;
           else next = READ;
    DONE : next = IDLE;
    default: next = XXX;
  endcase
end
```

Конечные автоматы. Петля в графе состояний

```
always_comb begin
  next = XXX;
  case (state)
    IDLE : if (go)  next = READ;
           else     next = IDLE; //@ loopback
    READ :          next = DLY;
    DLY  : if (!ws) next = DONE;
           else     next = READ;
    DONE :          next = IDLE;
    default:        next = XXX;
  endcase
end
```

next='x requires loopback state assignment

Loopback state to IDLE




```
always_comb begin
  next = state;
  case (state)
    IDLE : if (go)  next = READ;

    READ :          next = DLY;
    DLY  : if (!ws) next = DONE;
           else     next = READ;
    DONE :          next = IDLE;
    default:        next = XXX;
  endcase
end
```

next=state requires NO loopback state assignment

Конечные автоматы. Значения по умолчанию

```
always_ff @(posedge clk, negedge rst_n)
  if (!rst_n) begin
    rd <= '0;
    ds <= '0;
  end
  else begin
    rd <= '0;
    ds <= '0;
    case (next)
      IDLE : ;
      READ : rd <= '1;
      DLY  : rd <= '1;
      DONE : ds <= '1;
      default: {rd,ds} <= 'x;
    endcase
  end
endmodule
```

Конечные автоматы. Сравнение способов кодирования

prep4 Coding Goals	1 always	2 always	3 always	4 always
(1) Easy to change state encodings	yes	yes	yes	yes
(2) Concise coding style	no	yes	yes	yes
(3) Easy to code and understand	~no	yes	yes	~yes
(4) Facilitate debugging	yes	yes	yes	yes
(5) Yield efficient synthesis results	yes+	~no	~yes	yes+
(6) Easy to change due to FSM changes	~no	yes	yes	~yes

Упражнение: игра

1. Есть игровое устройство с 4 индикаторами на HEX.
2. На каждом циклически изменяется выводимое значение с счетчика.
3. Изменение счетчика останавливается при зажатии соответствующей кнопки. Для HEX0 это key[0]. Для HEX1 это key[1] и т.д.
4. Необходимо модифицировать упражнение добавив к нему конечный автомат. У упражнения есть несколько уровней сложности.

Упражнение: игра, первый уровень

1. Цель игры набрать на индикаторах последовательность цифр заданную в проекте. Например, 7489. Если игрок правильно набрал последовательность то он выиграл.
2. Изменение значений на индикаторе останавливается при нажатии на соответствующую кнопку. И не начинается заново до начала новой игры.
3. Остановка индикаторов возможно только в порядке от младшего к старшему. То есть если не остановился HEX0 нет реакции на key[1] key[2] key[3].
4. Как только останавливают все 4 индикатора то последовательность цифр сравнивается с этой которую должен был выбрать игрок и выводится сообщение о том проиграл или выиграл игрок.
5. Если при выводе результата игры на индикаторы нажать любую кнопку игра начнется заново.

Упражнение: игра, второй уровень

1. Измените логику работы конечного автомата так чтобы неправильная последовательность нажатия кнопок приводила к завершению игры. То есть если нажать key[1] раньше key[0]. Игра завершится поражением.

Упражнение: игра, третий уровень

1. Добавьте дополнительные модули генерации стробов для того чтобы скорость изменения индикаторов была разной.
2. Чем старше номер индикатора тем быстрее меняются цифры.

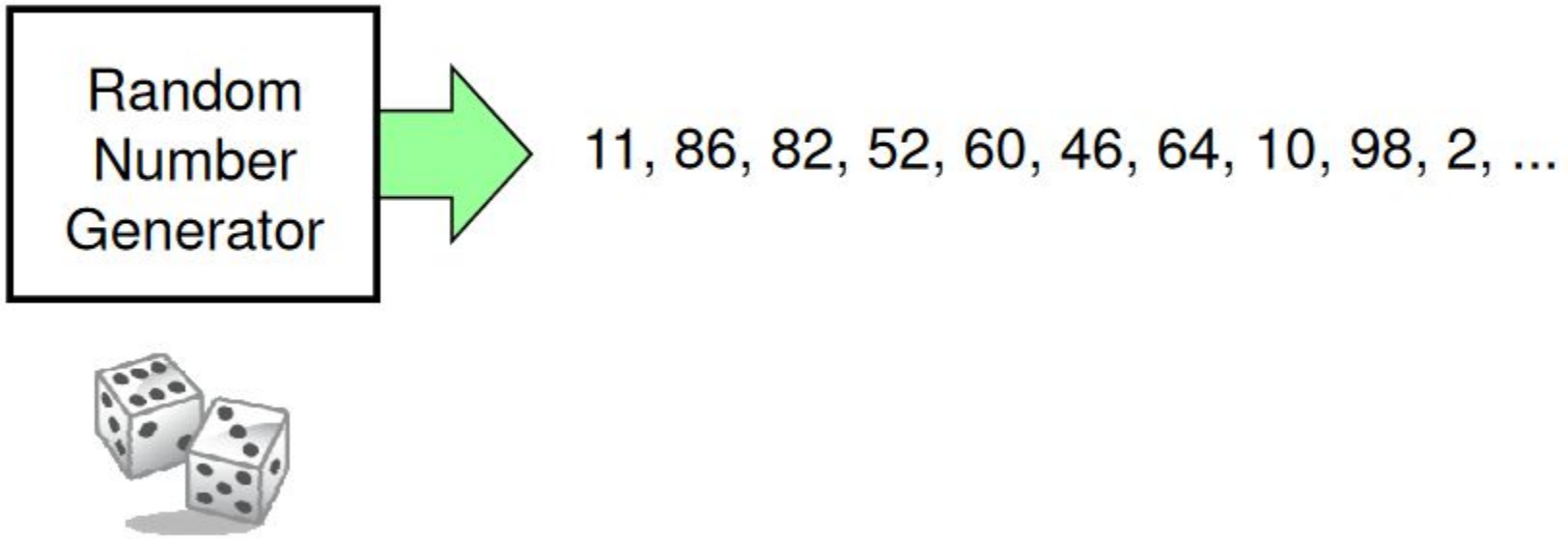
Упражнение: игра, четвёртый уровень

1. Хорошо если бы цифры менялись на индикаторах не последовательно а случайно.
2. Модифицируйте модуль счетчика так чтобы значения из него генерировались по псевдослучайной последовательности.

Генерация псевдослучайной последовательности чисел в RTL

1. Материалы для это части презентации взяты из лекции ***Patrick Schaumont ECE 4514 Digital Design II Lecture 6: A Random Number Generator in Verilog***
2. В прошлом году про генератор чисел читал лекцию Илья Кудрявцев из Самарского университета. Очень интересно, посмотрите. https://youtu.be/8lYVOb2JZOU?t=7622

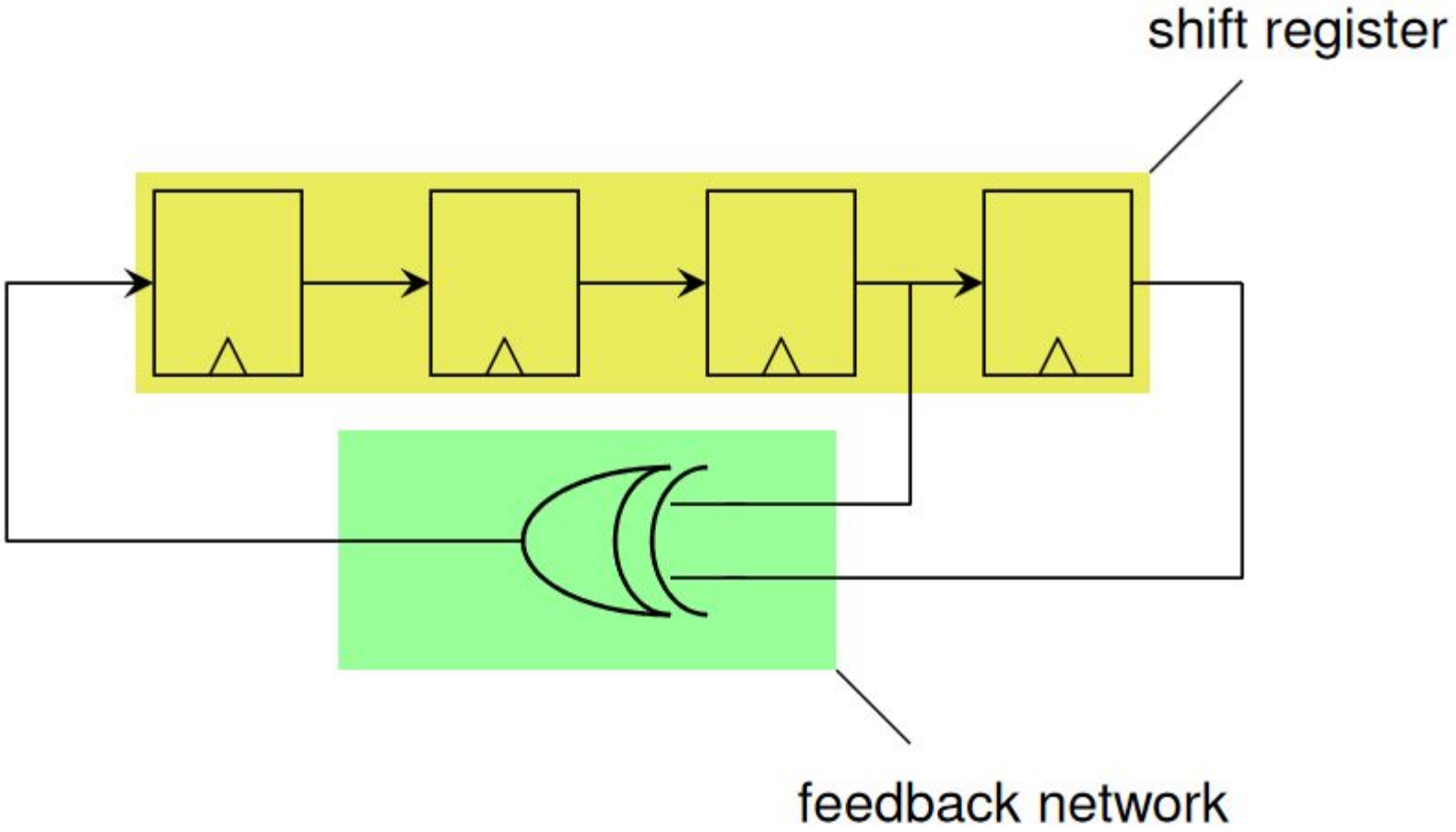
Генерация псевдослучайной последовательности чисел в RTL



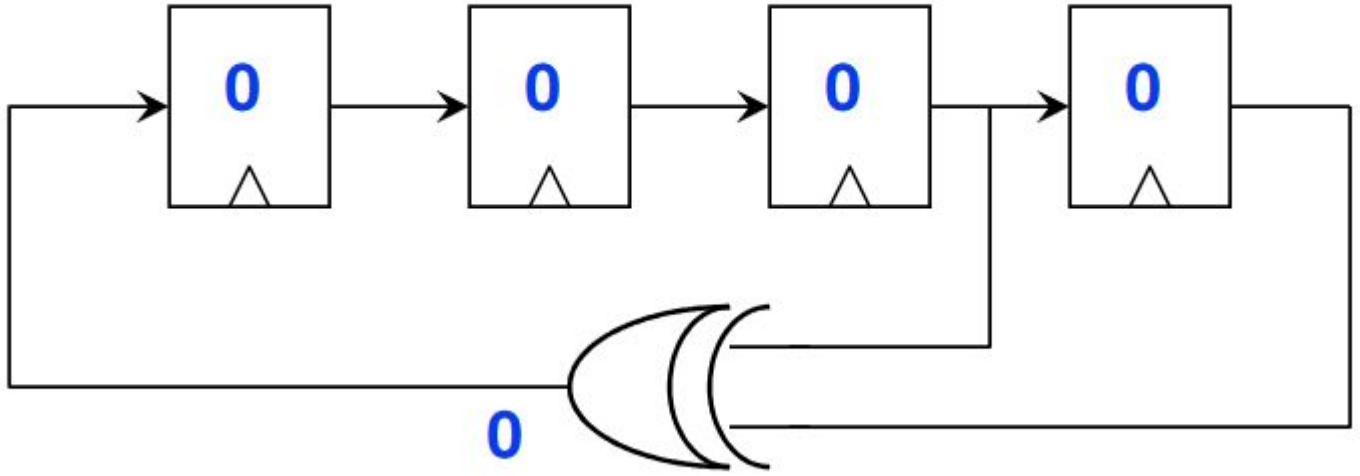
Генерация псевдослучайной последовательности чисел в RTL

```
module random(q);
  output [0:31] q;
  reg [0:31] q;

  initial
    r_seed = 2;

  always
    #10 q = $random(r_seed);
endmodule
```

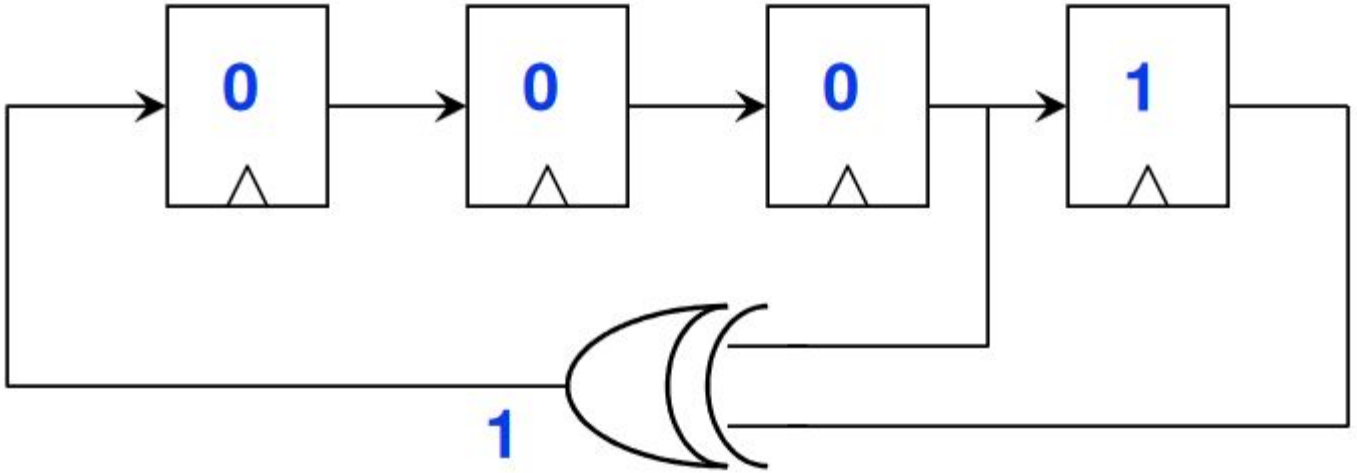
Linear Feedback Shift Register



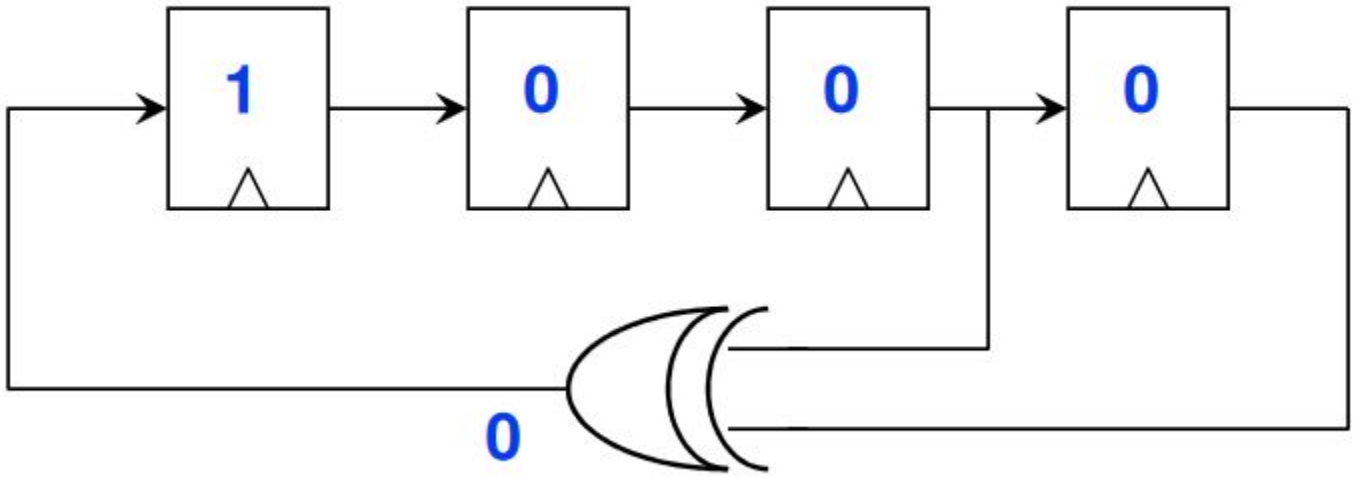
Linear Feedback Shift Register



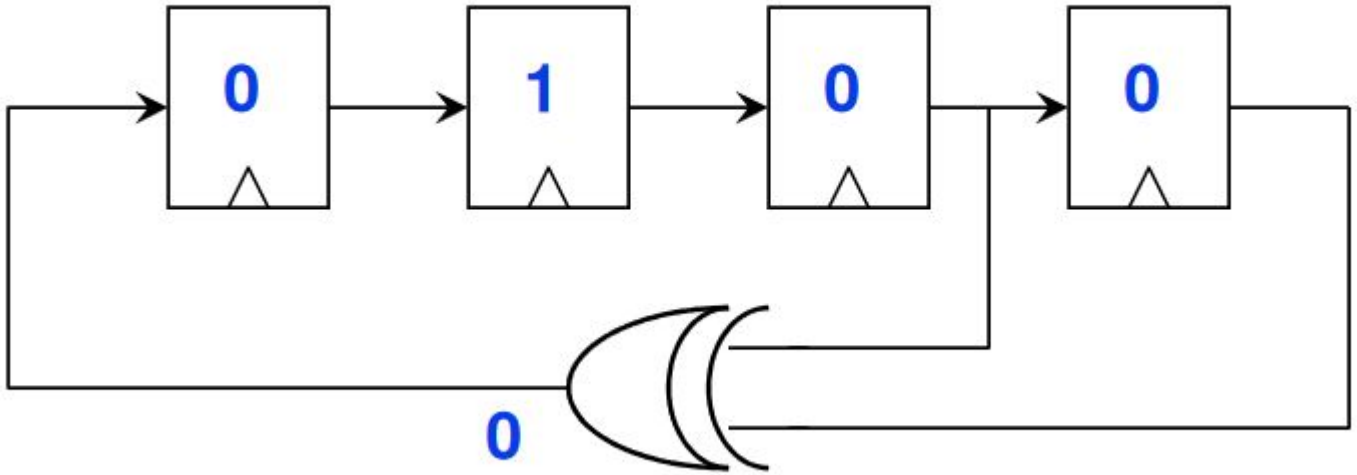
Linear Feedback Shift Register



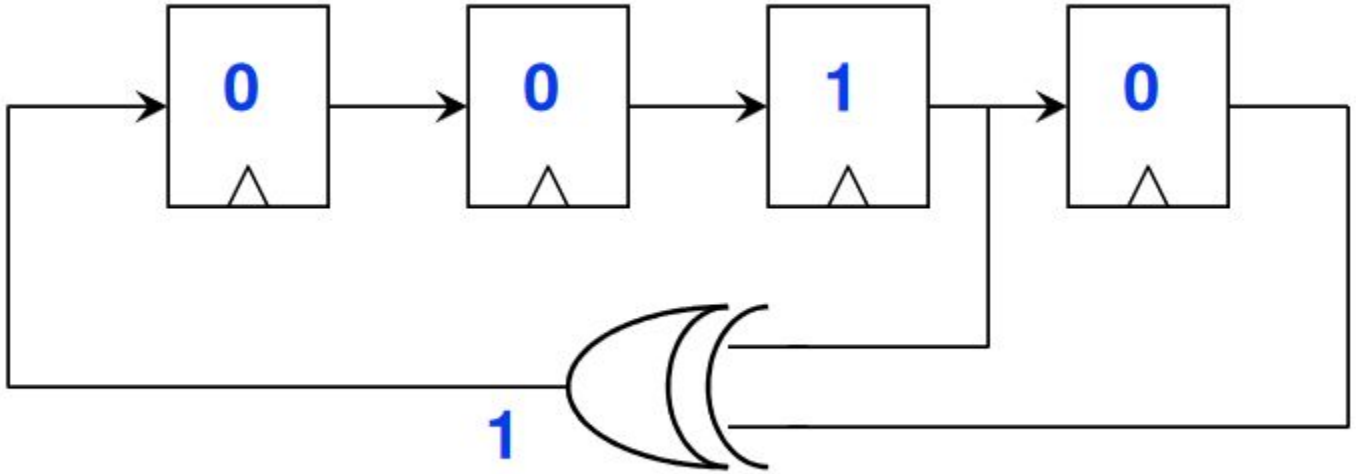
Linear Feedback Shift Register



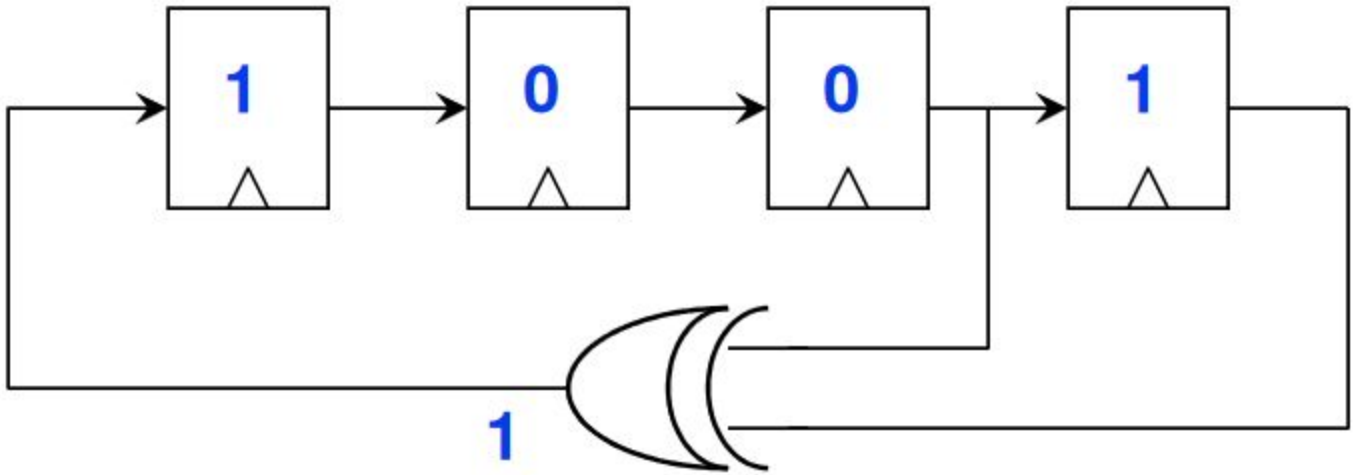
Linear Feedback Shift Register



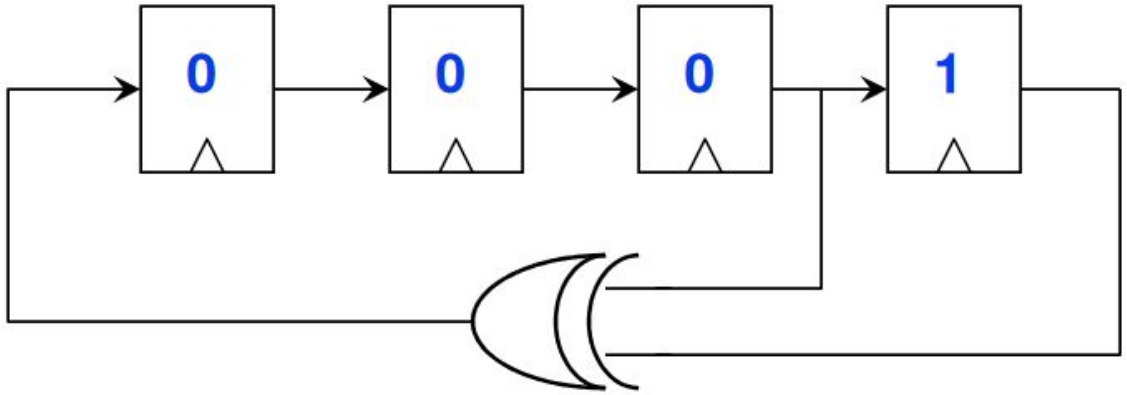
Linear Feedback Shift Register

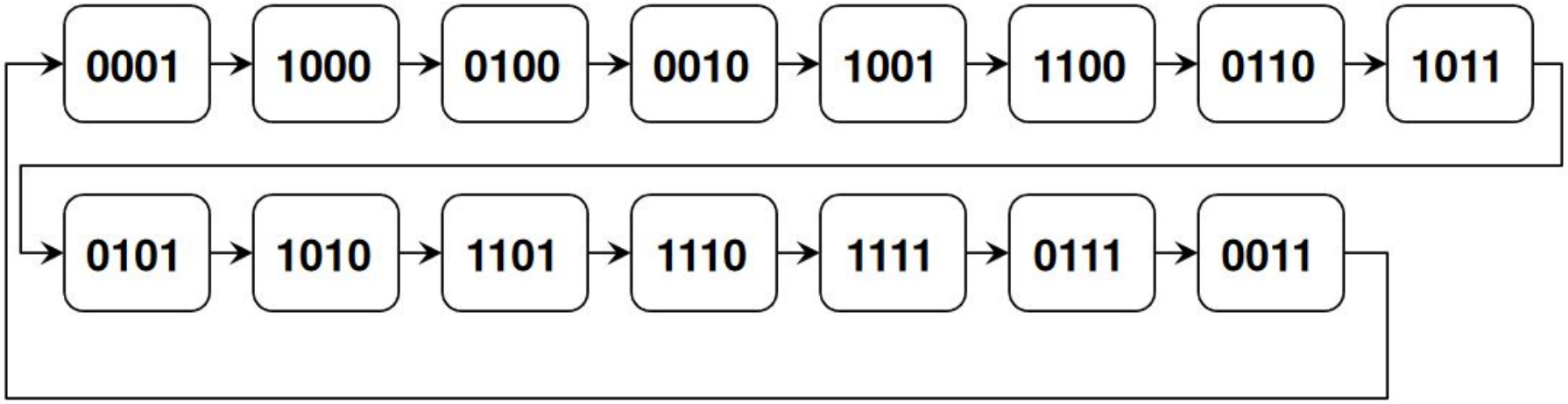


Linear Feedback Shift Register

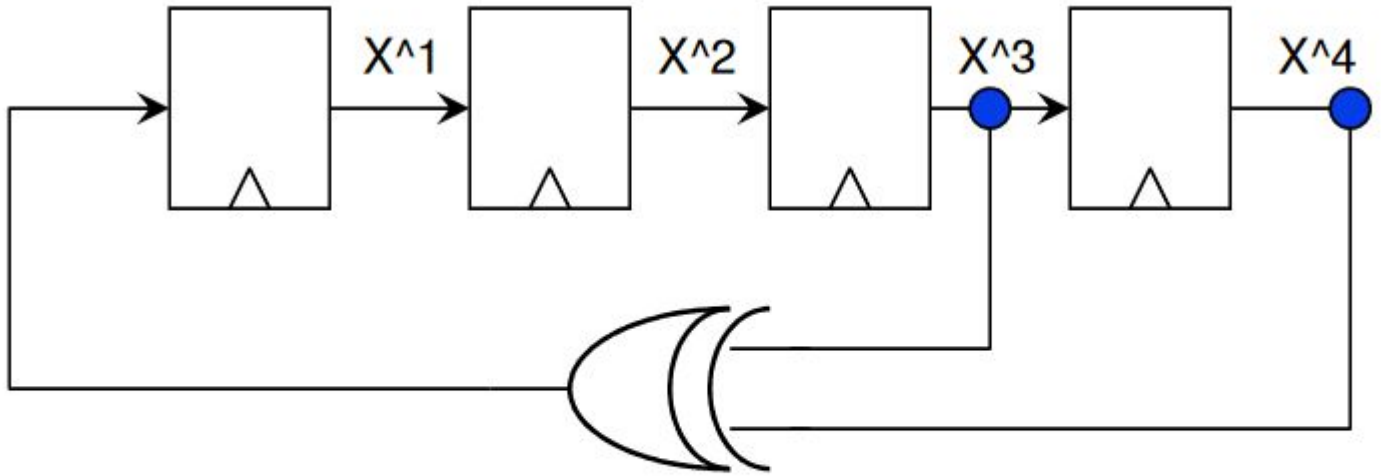


Linear Feedback Shift Register





Linear Feedback Shift Register



$$P(x) = x^4 + x^3 + 1$$

Спасибо за внимание.