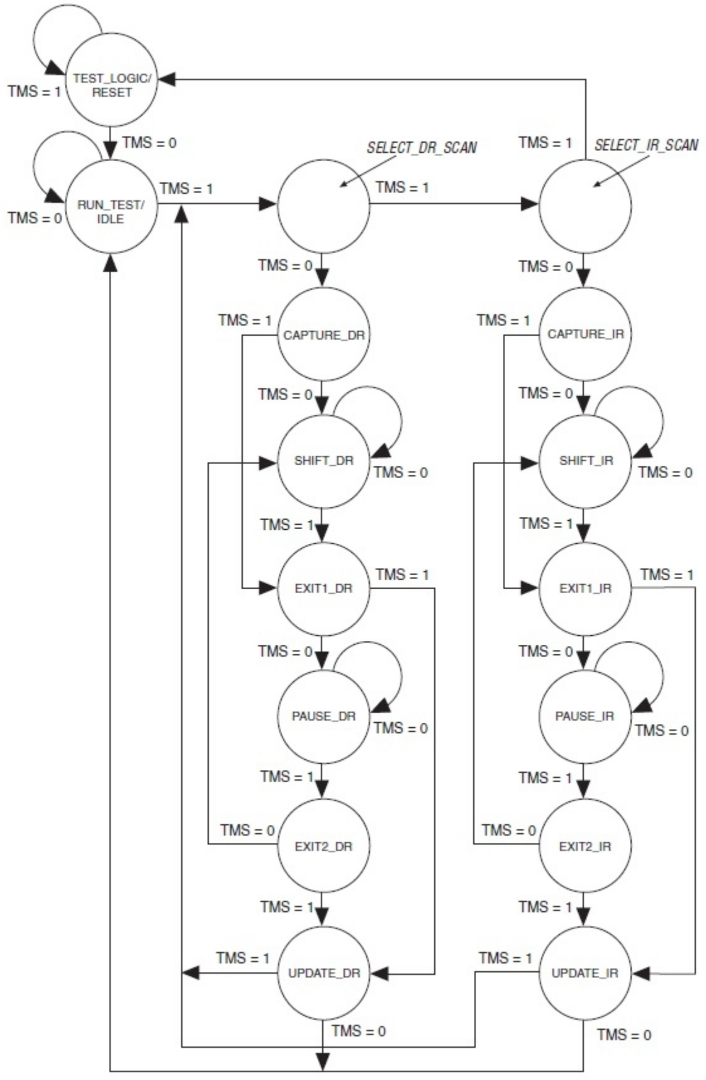
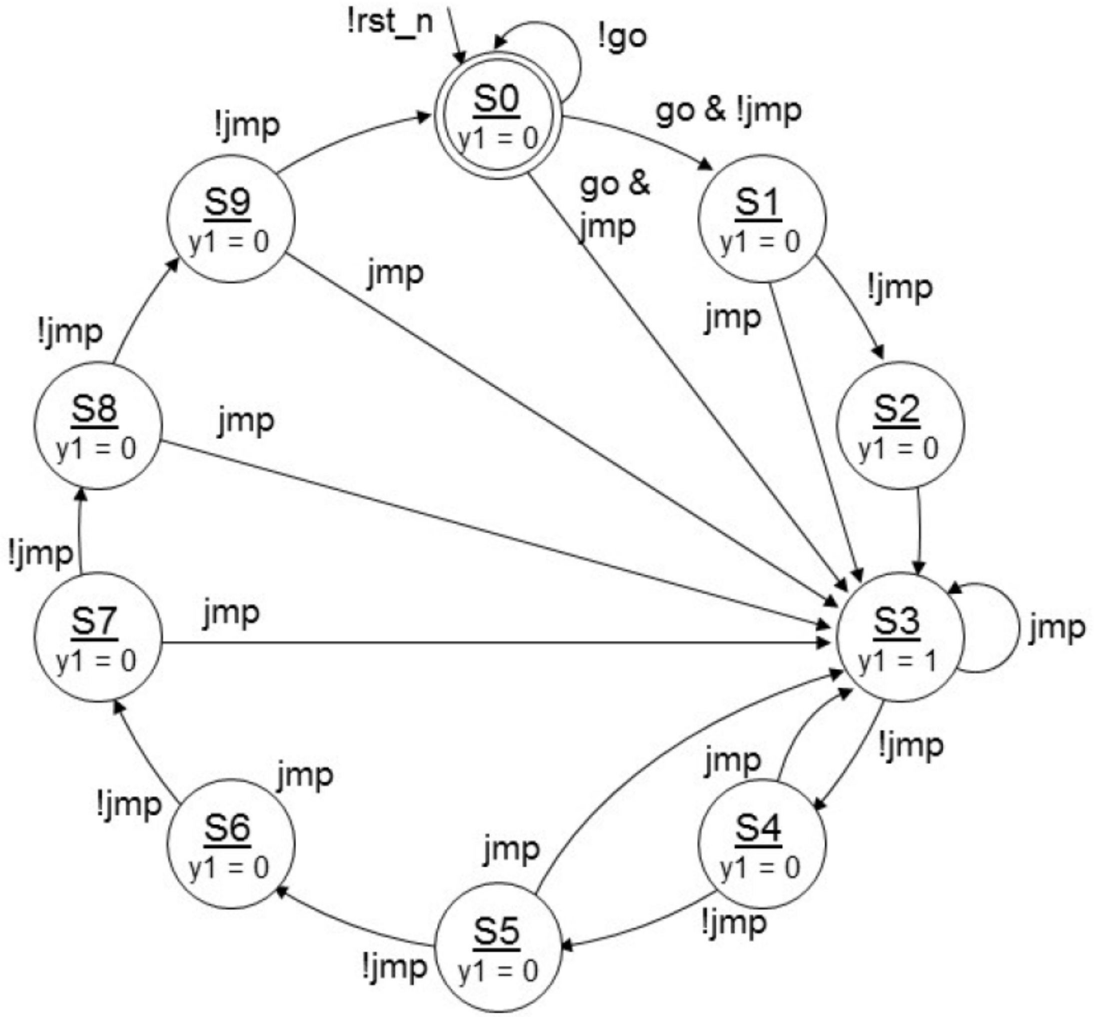
Finite state machines

Использование

- Любые системы с последовательными переходами
- Интерфейсы
- Тестирование

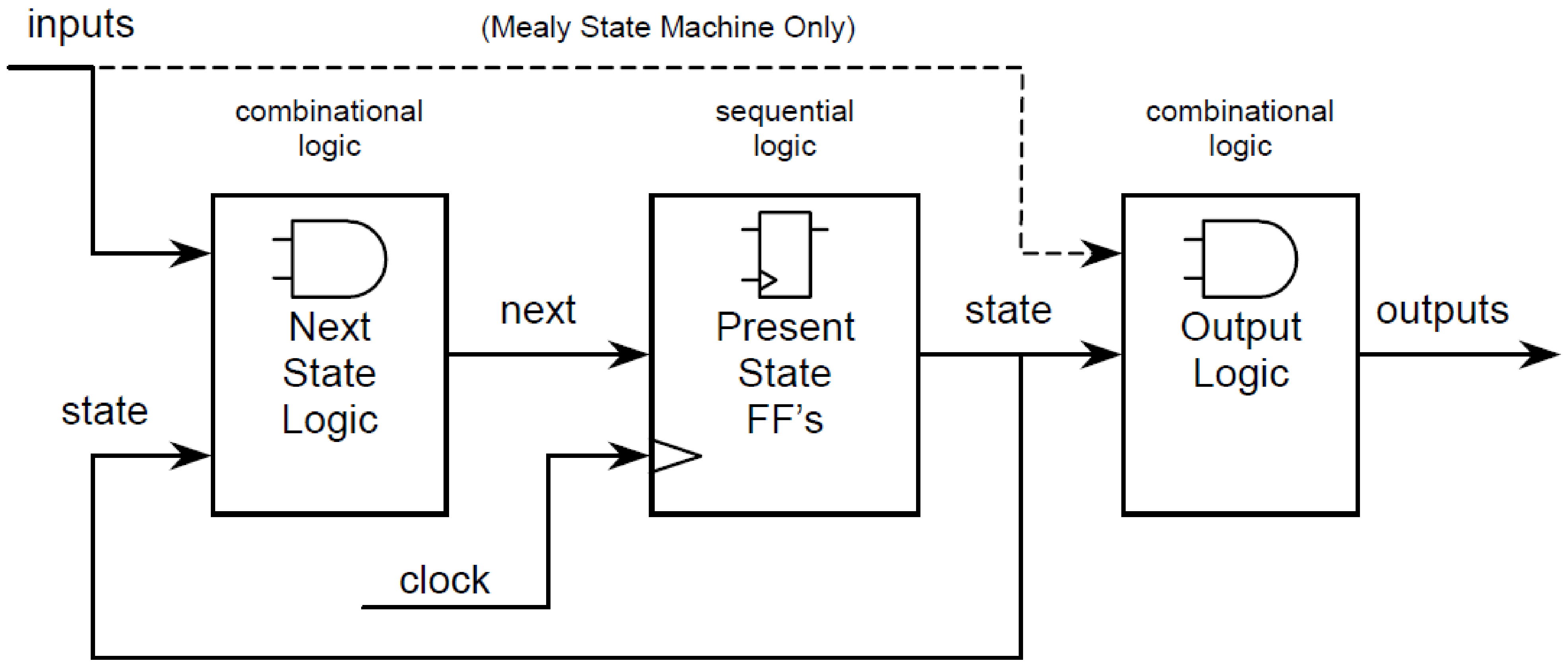
Описание графом переходов





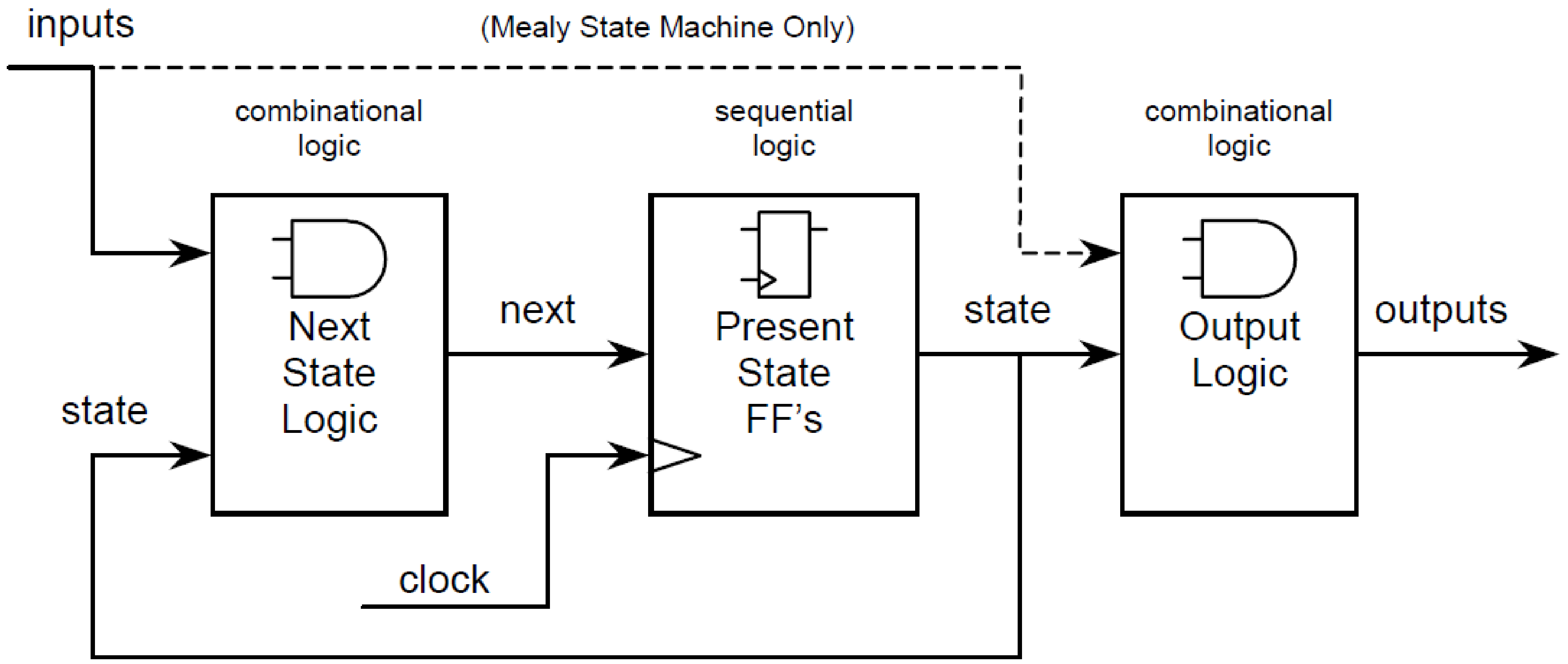
Moore type

- Выходы есть функции только текущего состояния
- Есть целый такт на распространение сигнала по комбинационному пути

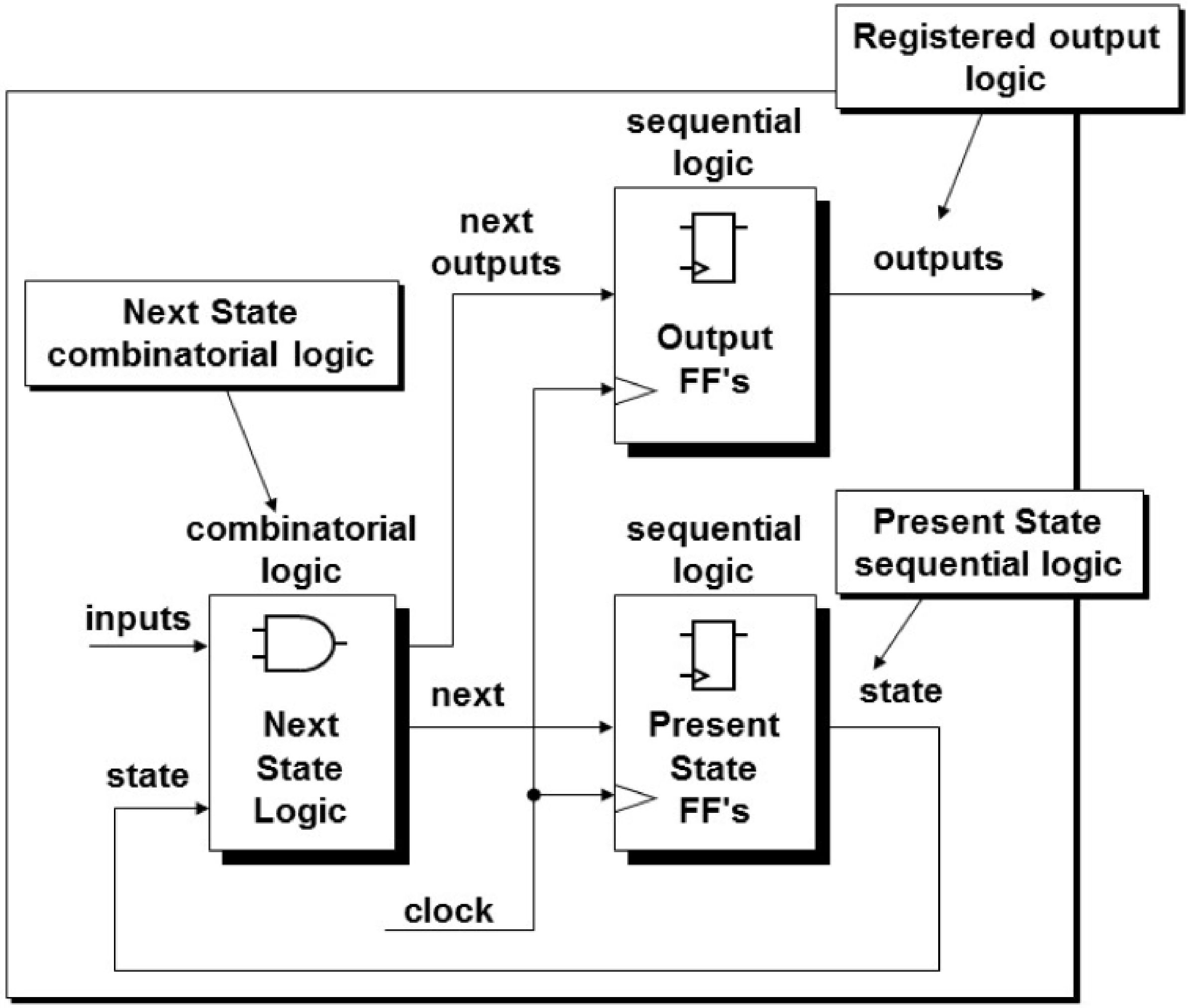


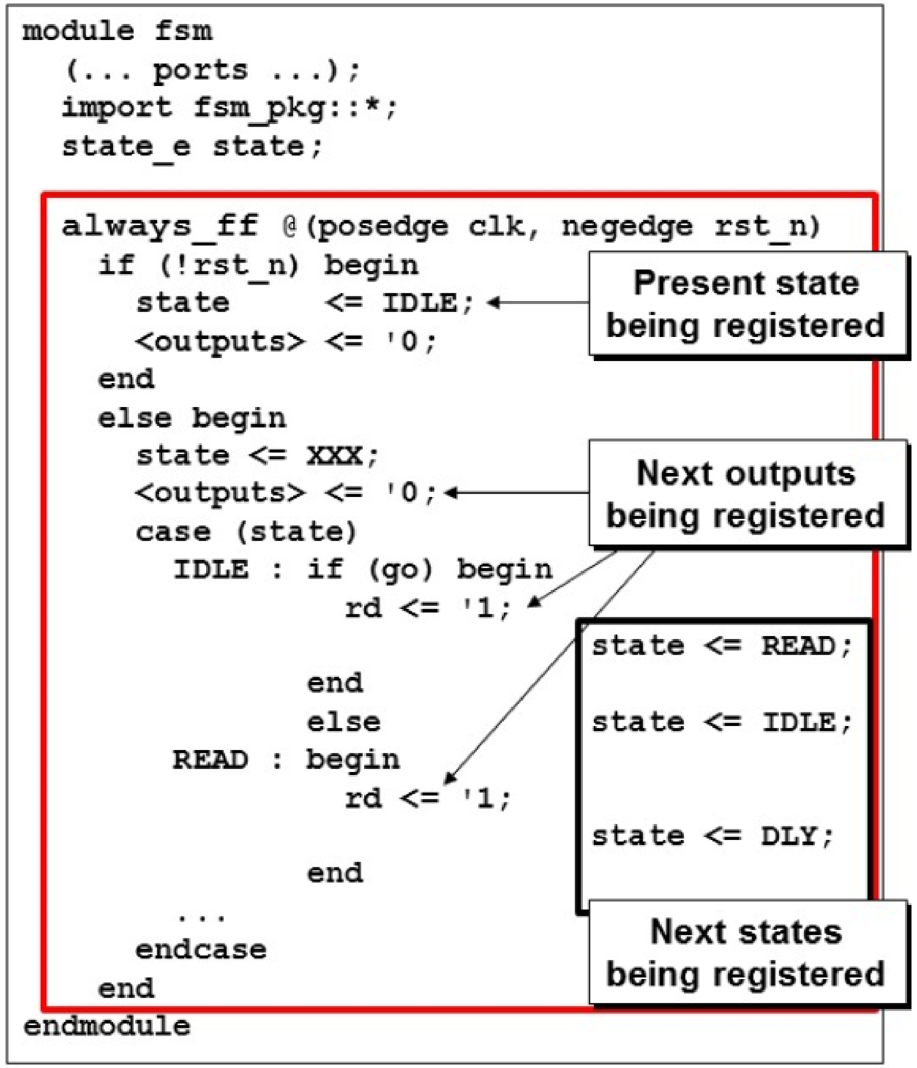
Mealy type

- Часть выходов есть функции только текущего состояния и часть выходов есть функции входов
- Есть условия на тайминг входных сигналов



One always block with registered outputs

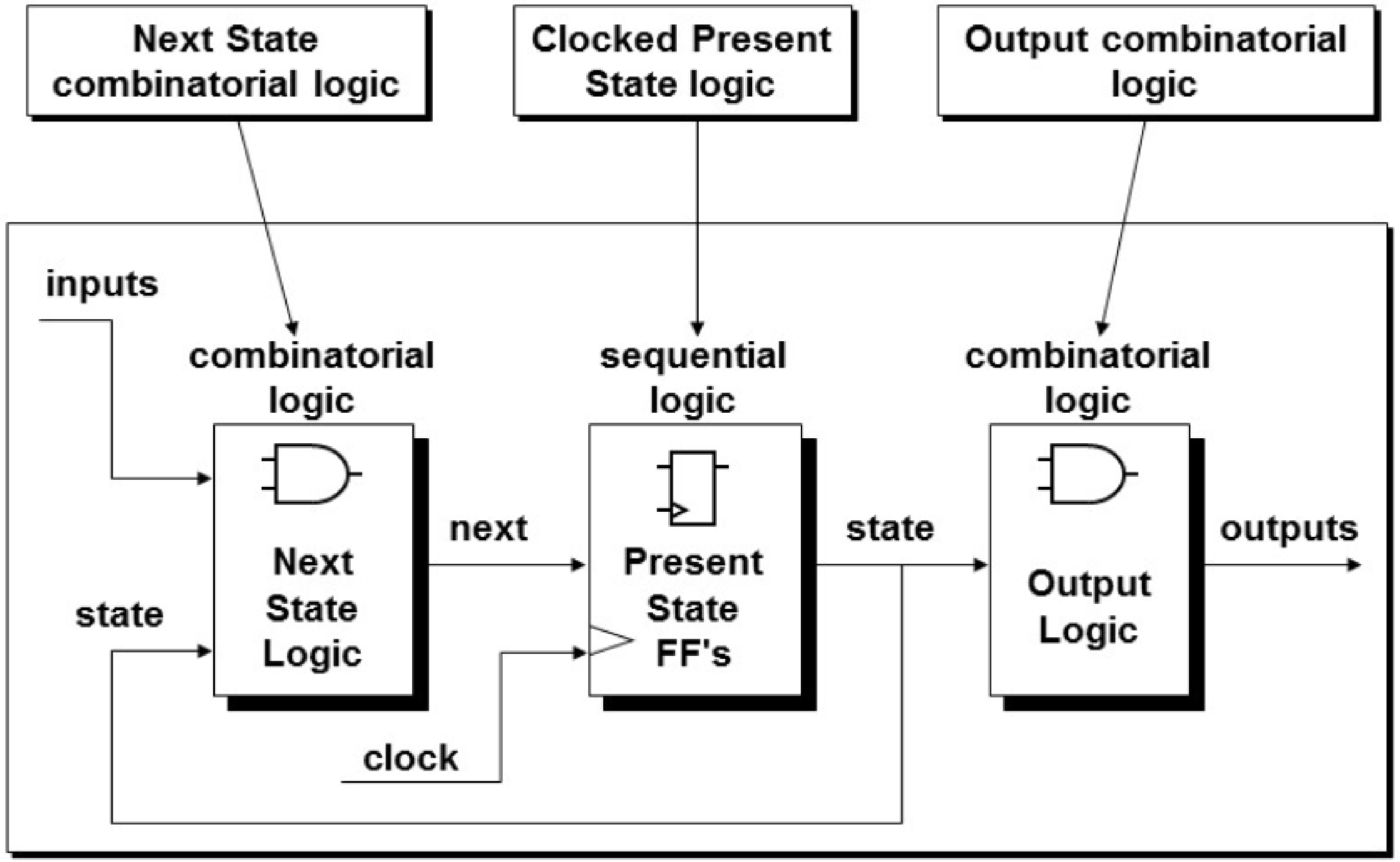


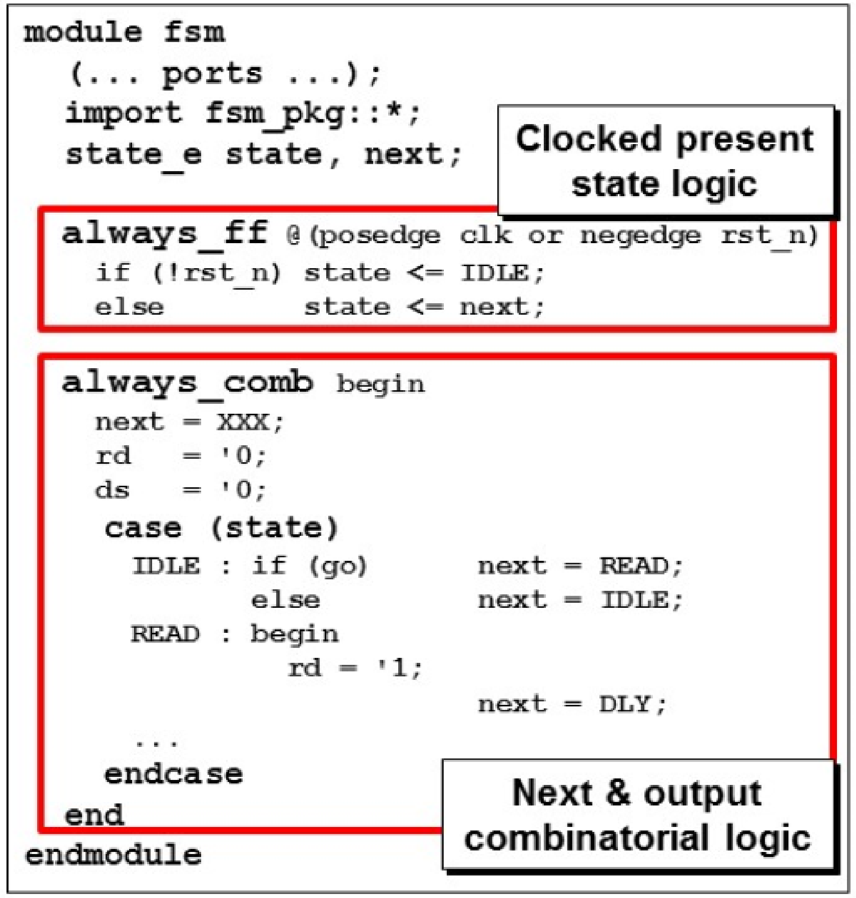


```
module fsm
  (... ports ...);
  import fsm_pkg::*;
  state_e state;

  always_ff @(posedge clk, negedge rst_n)
    if (!rst_n) begin
      state     <= IDLE;
      <outputs> <= '0;
    end
    else begin
      state <= XXX;
      <outputs> <= '0;
      case (state)
        IDLE : if (go) begin
                 rd <= '1;
                                       state <= READ;
               end
               else                    state <= IDLE;
        READ : begin
                 rd <= '1;
                                       state <= DLY;
               end
        ...
      endcase
    end
endmodule
```

Present state being registered
Next outputs being registered
Next states being registered

Two always blocks, combinational outputs

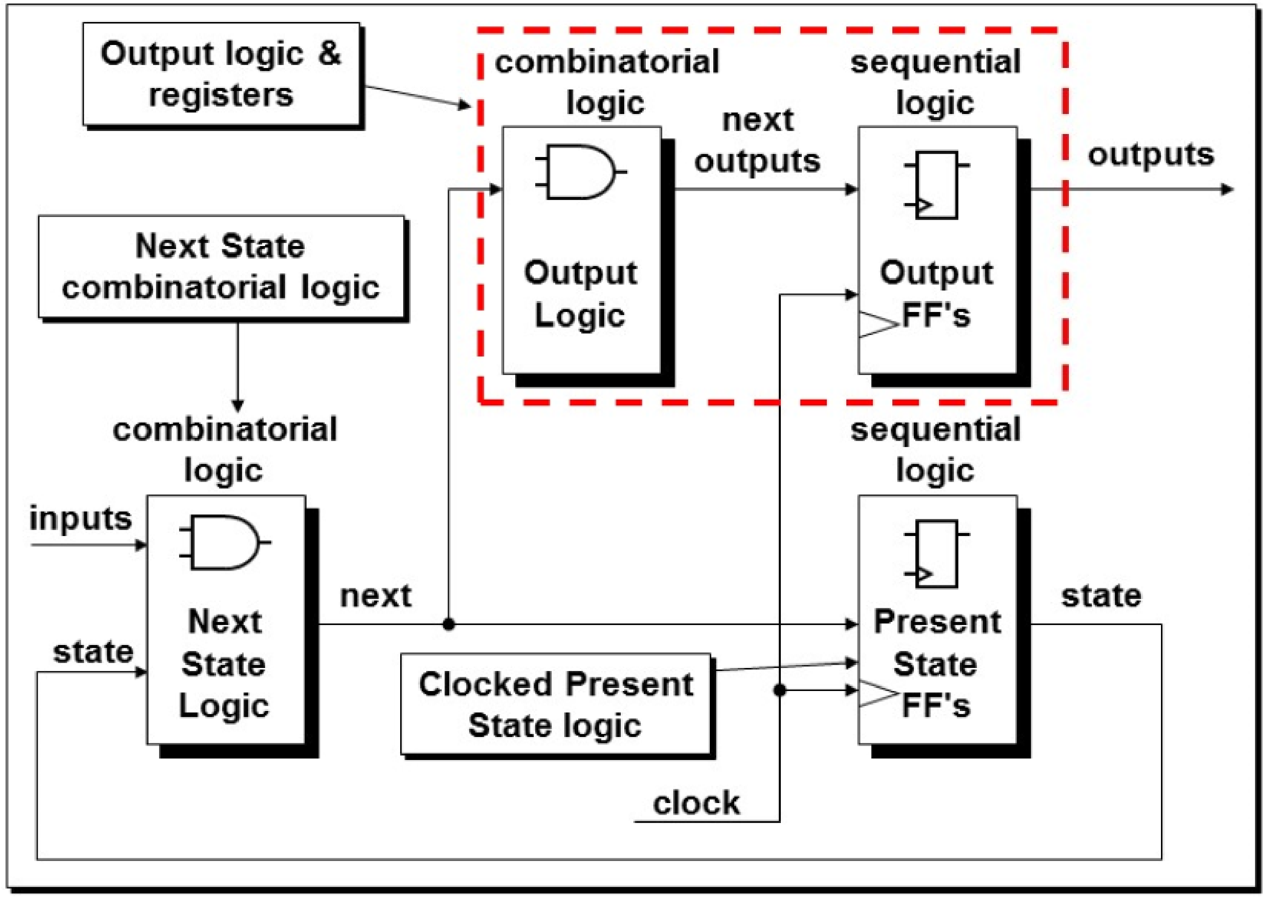


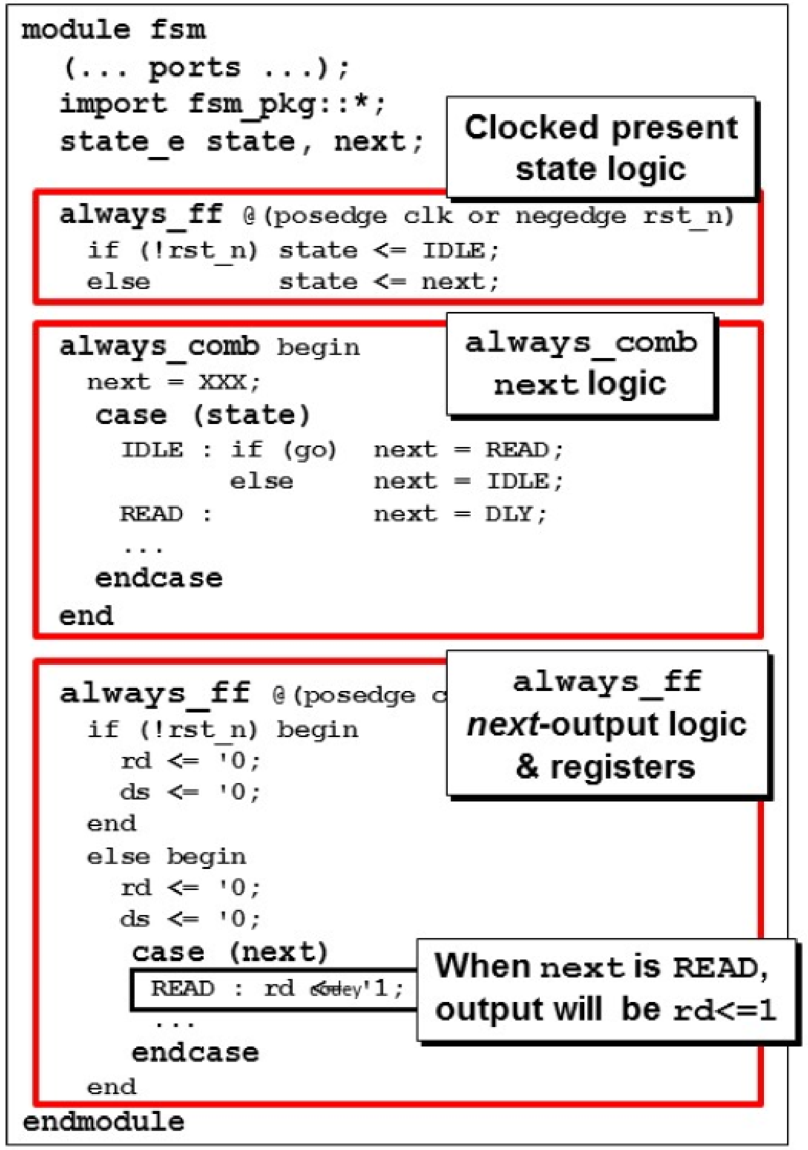


Clocked present state logic
Next & output combinatorial logic

```
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
```

Three always blocks, registered outputs





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

Clocked present state logic

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

always_comb next logic

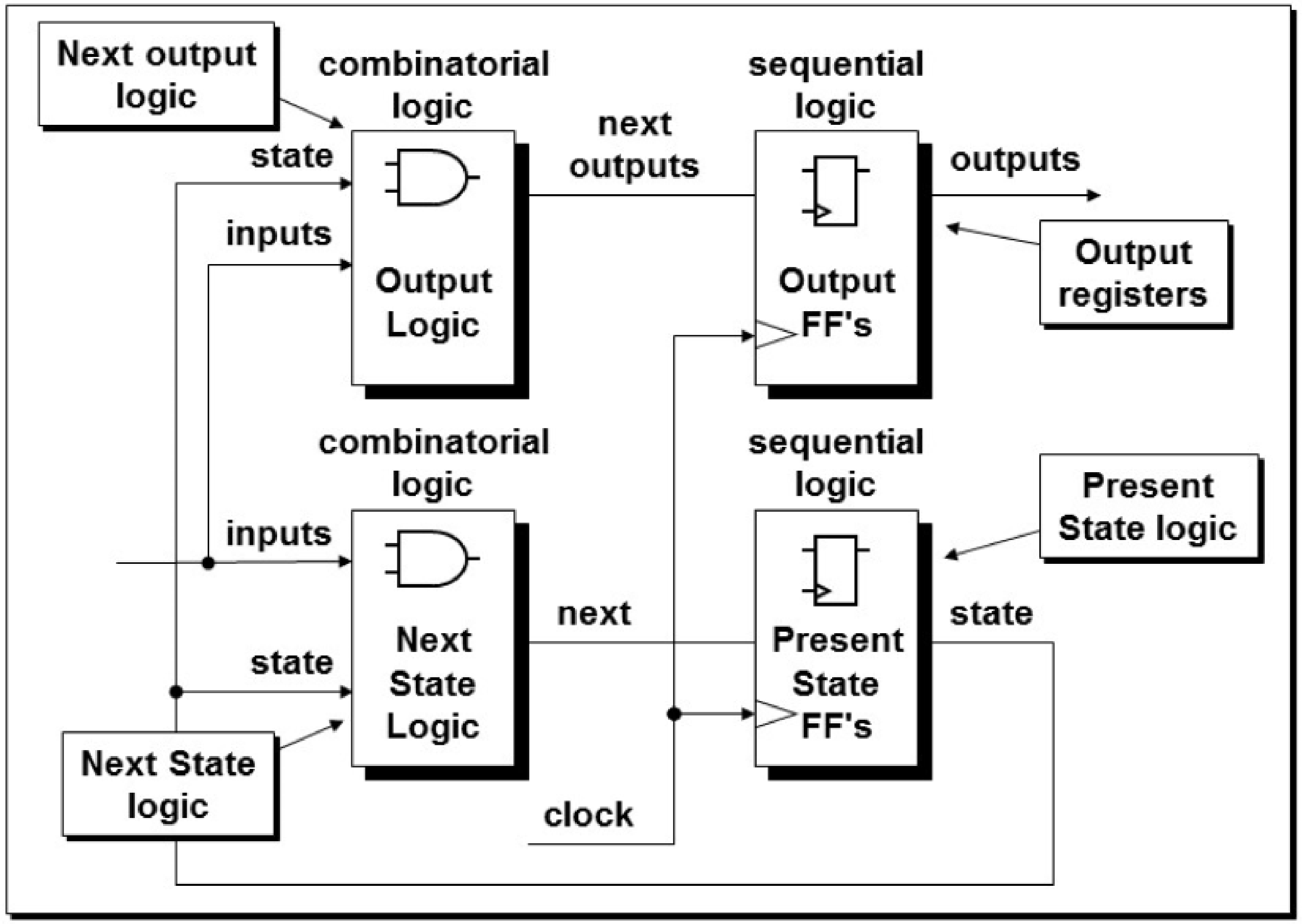
```
  always_comb begin
    next = XXX;
    case (state)
      IDLE : if (go)   next = READ;
             else      next = IDLE;
      READ :           next = DLY;
      ...
    endcase
  end
```

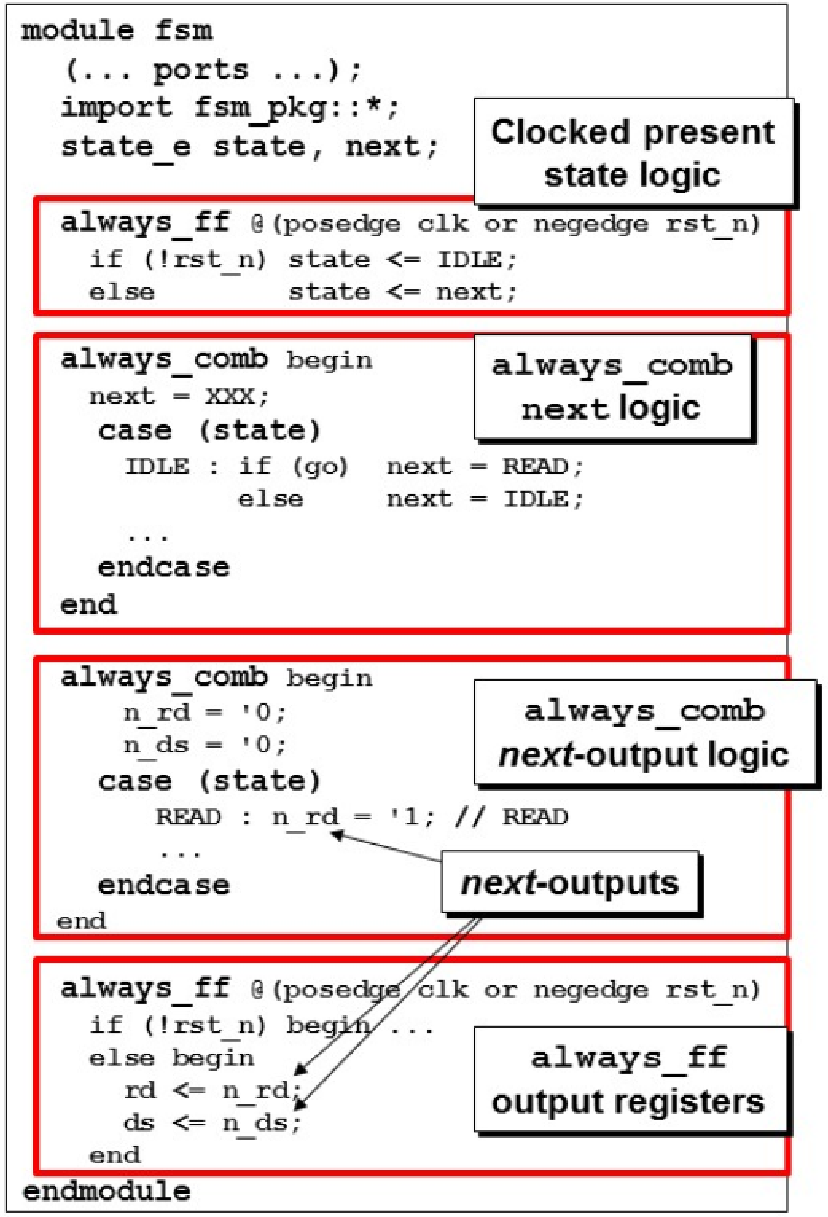
always_ff *next*-output logic & registers

```
  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
```

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

Four always blocks, registered outputs

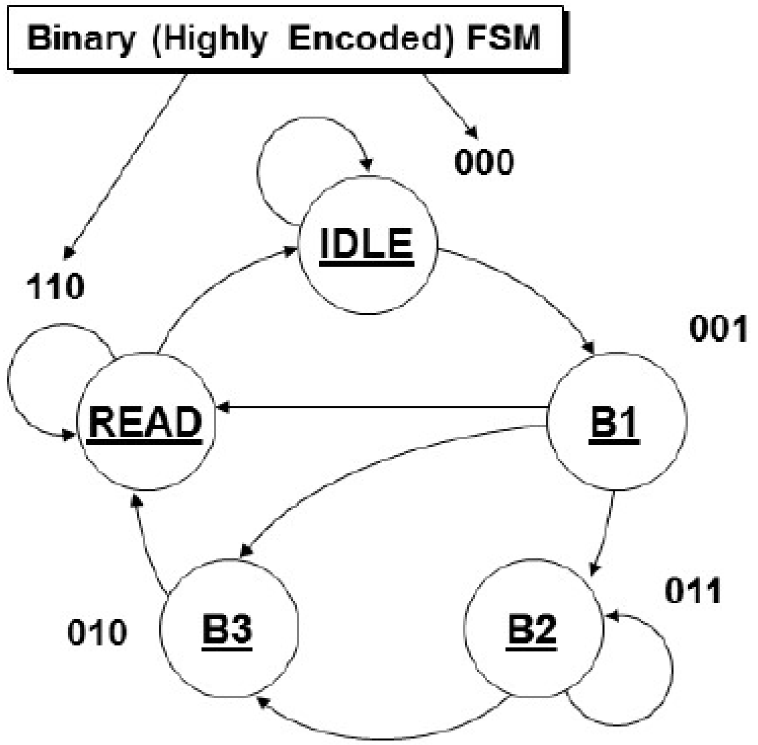


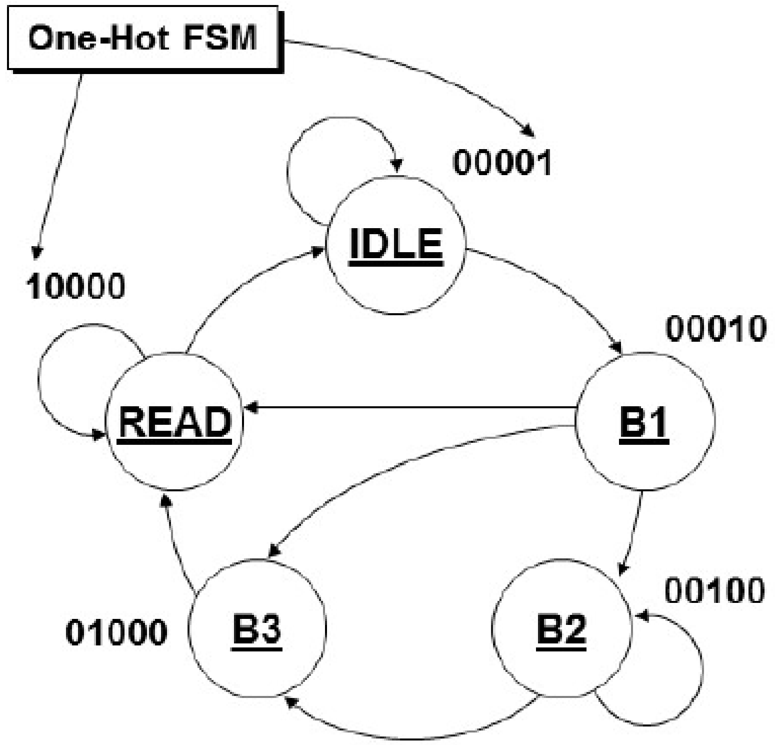

Clocked present state logic
always_comb next logic
always_comb *next*-output logic
next-outputs
always_ff output registers

```
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
```

Binary vs OneHot encoding

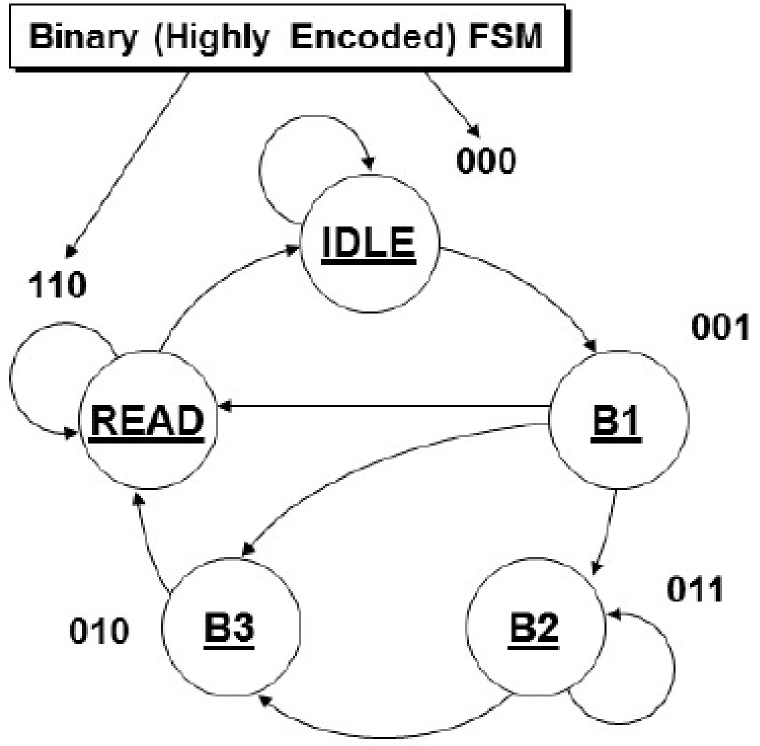
- One-hot быстрее переключается между состояниями, но требует больше всего триггеров

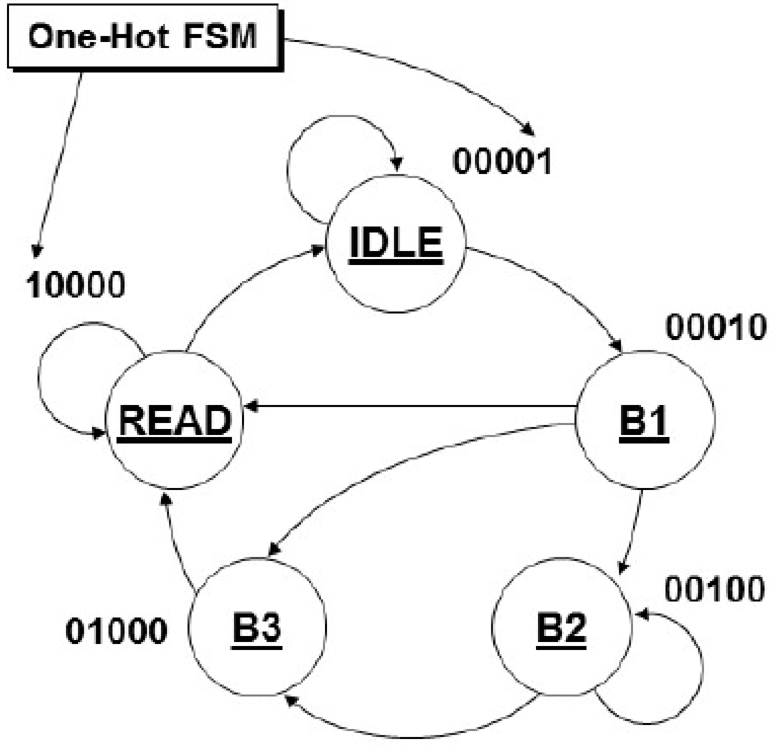




ASIC -vs- FPGA Synthesis

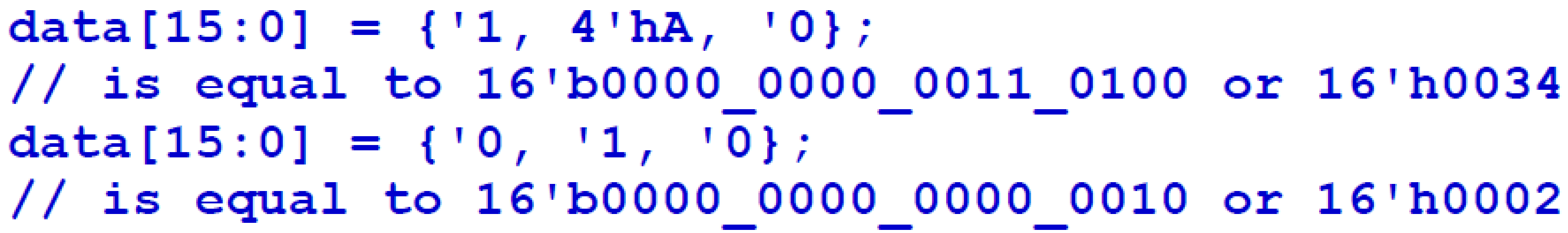
- ASIC: “больше” комбинационных элементов
- FPGA: “больше” последовательностных элементов





Code style and safety

- localparam
- case, casex, casez
- default state
- All zero, all ones, all x



```
data[15:0] = {'1, 4'hA, '0};
// is equal to 16'b0000_0000_0011_0100 or 16'h0034
data[15:0] = {'0, '1, '0};
// is equal to 16'b0000_0000_0000_0010 or 16'h0002
```

- Use XXX state to debug

FSM state register

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

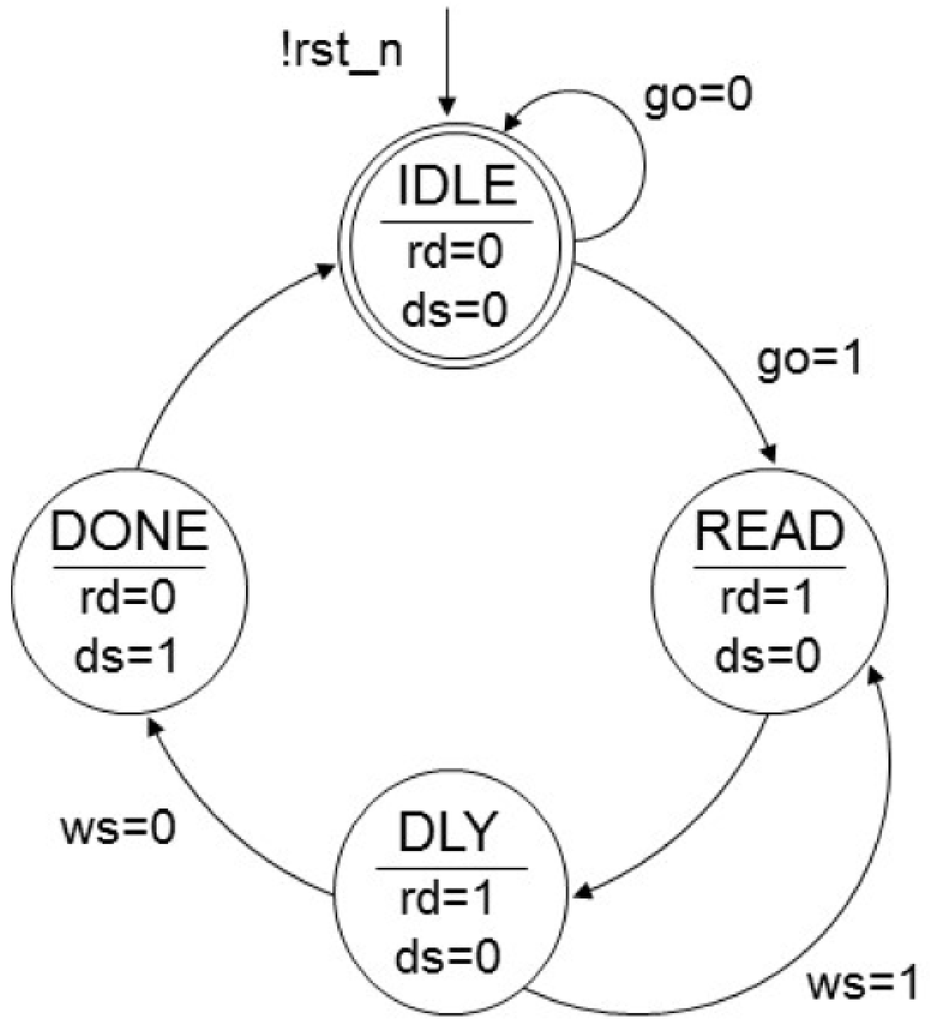
```
always_ff @(posedge clk, negedge rst_n) begin
  testbad <= go;
  if (!rst_n) state <= IDLE;
  else        state <= next;
end
```

Use pre-default-x

- Ошибка в случае потери состояния
- Явное переключение состояния
- Лучше результаты синтеза

```
always_comb begin
  next = XXX;                      // Pre-default-X assignment
  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;  // case-default-X assignment
  endcase
end
```

Пример


!rst_n
go=0
IDLE
rd=0
ds=0
go=1
READ
rd=1
ds=0
DLY
rd=1
ds=0
ws=1
ws=0
DONE
rd=0
ds=1

Пример

```
always_ff @(posedge clk, negedge rst_n)
  if (!rst_n) begin
    state <= IDLE;
    rd    <= '0;
    ds    <= '0;
  end
  else begin
    state <= XXX;                          //@ LB
    rd    <= '0;
    ds    <= '0;
    case (state)
      IDLE : if (go) begin
               rd <= '1;
                            state <= READ;
             end
             else           state <= IDLE; //@ LB
      READ : begin
               rd <= '1;
                            state <= DLY;
             end
```

```
      DLY  : if (!ws) begin
               ds <= '1;
                            state <= DONE;
             end
             else begin
               rd <= '1;
                            state <= READ;
             end
      DONE :                state <= IDLE;
      default: begin
               ds <= 'x;
               rd <= 'x;
                            state <= XXX;
             end
    endcase
  end
```

Пример

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

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

```
    DLY  : begin
             rd = '1;
             if (!ws) next = DONE;
             else     next = READ;
           end
    DONE : begin
             ds = '1;
                      next = IDLE;
           end
    default: begin
             ds = 'x;
             rd = 'x;
                      next = XXX;
           end
  endcase
end
```

Пример

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

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

```
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
```

Пример

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

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

```
always_comb begin
  n_rd = '0;
  n_ds = '0;
  case (state)
    IDLE : if ( go)  n_rd = '1; // READ
           else ;               // IDLE
    READ :           n_rd = '1; // DLY
    DLY  : if ( ws)  n_rd = '1; // READ
           else      n_ds = '1; // DONE
    DONE : ;                    // IDLE
    default:        {n_rd,n_ds} = 'x;
  endcase
end

always_ff @(posedge clk, negedge rst_n)
  if (!rst_n) begin
    rd <= '0;
    ds <= '0;
  end
  else begin
    rd <= n_rd;
    ds <= n_ds;
  end
```

Литература

Sunburst Design -
FiniteStateMachine(FSM)Design&SynthesisusingSystemVerilog-PartI