Крайни Автомати -

Finite State Machines (FSM)

[Типове Крайни Автомати](#_jib9p1e7cn0y)

[Кодиране на състоянията](#_rxt817ph91lr)

[Диаграма на състоянията (State Diagram)](#_5yn6mjk78i6g)

[ASM (Algorithmic State Мachine) Диаграма](#_ix1411d1qwz8)

[Описание на крайни автомати с Verilog](#_piiav9jhiu7o)

[Два always блока](#_ezmncmrxwmlz)

[Описание на състоянията с изброим тип](#_pnkic74pfvod)

[Три always блока с регистрови изходи](#_62podar61w95)

[Пример - fsm\_cc4\_2](#_cqn5rxr45f92)

[Combinational outputs](#_4v0axjvr8opl)

[Registered outputs](#_269t1q9ra6cx)

[Включване на изходите в кодирането на състоянията](#_p9x8qdmiirg9)

[Литература](#_9m7byumrq3um)

Приложения на крайните автомати в цифровите системи



# Типове Крайни Автомати



Moore: изходите зависят само от текущото състояние

Mealy: изходите зависят от текущото състояние и входовете

* Текущото състояние се съхранява в D тригери
* Следващото състояние на автоматът се определя от текущото състояние и входните сигнали
* Определянето на следващото състояние се извършва от комбинационна логическа схема

## Кодиране на състоянията

Binary : 000, 001, 010, 011, 100, 101,....  
Grey : 000, 001, 011, 010, 110, 100,....  
One Hot : 000001, 000010, 000100, 001000, 010000,....  
Random : 001000, 100010, 000110, 101000, 110110,....

## Диаграма на състоянията (State Diagram)



State machine diagrams document how the state machine operates

* States are given names inside the bubbles
* Transitions between states are indicated by arcs
* Conditions for taking the arc given by a logic equation
* Outputs asserted in states are given by a list beside the state
* Mealy outputs formed are described by a separate equation

## ASM (Algorithmic State Мachine) Диаграма



x - Moore output

z - Mealy output

bc - Boolean condition

# Описание на крайни автомати с Verilog

## Два always блока



| module fsm1(  output logic rd, ds,  input go, ws, clock, reset\_n  );  // state type  typedef enum logic [1:0] {IDLE, READ, DELAY, DONE} state\_t;  state\_t state, state\_next;  // state register  always\_ff @(posedge clock, negedge reset\_n)  if (~reset\_n) state <= IDLE;  else state <= state\_next;  // next state and output logic  always\_comb begin  **state\_next = state; // default next state**  **ds = '0; // default output values**  **rd = '0;**  case (state)  IDLE:  if (go) state\_next = READ;  READ: begin  rd = '1;  state\_next = DELAY;  end  DELAY: begin  rd = '1;  if (ws) state\_next = READ;  else state\_next = DONE;  end  DONE: begin  ds = '1;  state\_next = IDLE;  end  endcase  end  endmodule | **always\_ff**  блок описва регистъра на състоянията.  **alyaws\_com** блок описва преходите между състоянията.  **always\_comb** блока започва с установяване на следващо състояние по подразбиране  пак там се присвояват и стойности на изходите по подразбиране  в състоянията, където стойностите на изходите се различават от тези по подразбиране, присвояването се прави само веднъж. |
| --- | --- |

run -all

0.00 ns - go=0 ws=0: state=IDLE(0): rd=0 ds=0

200.00 ns - go=1 ws=0: state=IDLE(0): rd=0 ds=0

250.00 ns - go=1 ws=0: state=READ(1): rd=1 ds=0

350.00 ns - go=1 ws=0: state=DELAY(2): rd=1 ds=0

400.00 ns - go=0 ws=1: state=DELAY(2): rd=1 ds=0

450.00 ns - go=0 ws=1: state=READ(1): rd=1 ds=0

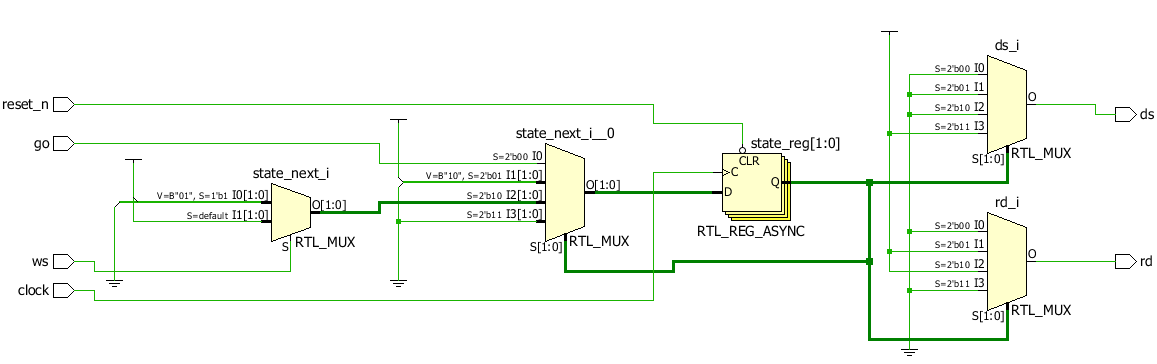
550.00 ns - go=0 ws=1: state=DELAY(2): rd=1 ds=0

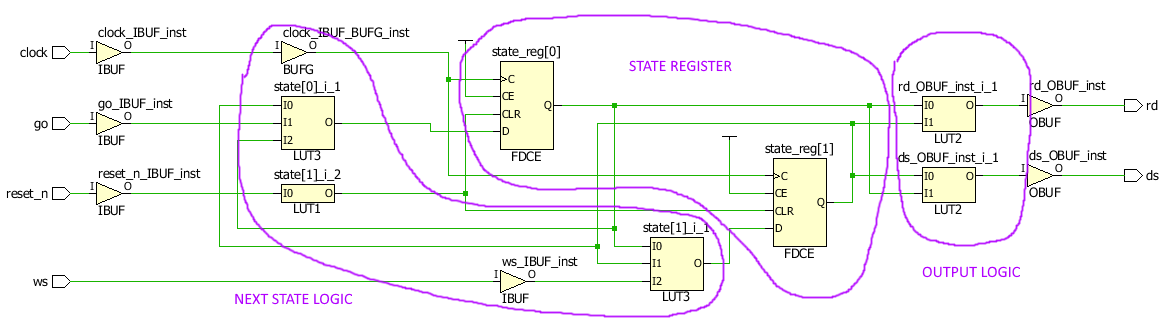
600.00 ns - go=0 ws=0: state=DELAY(2): rd=1 ds=0

650.00 ns - go=0 ws=0: state=DONE(3): rd=0 ds=1

750.00 ns - go=0 ws=0: state=IDLE(0): rd=0 ds=0

$finish called at time : 1 us :

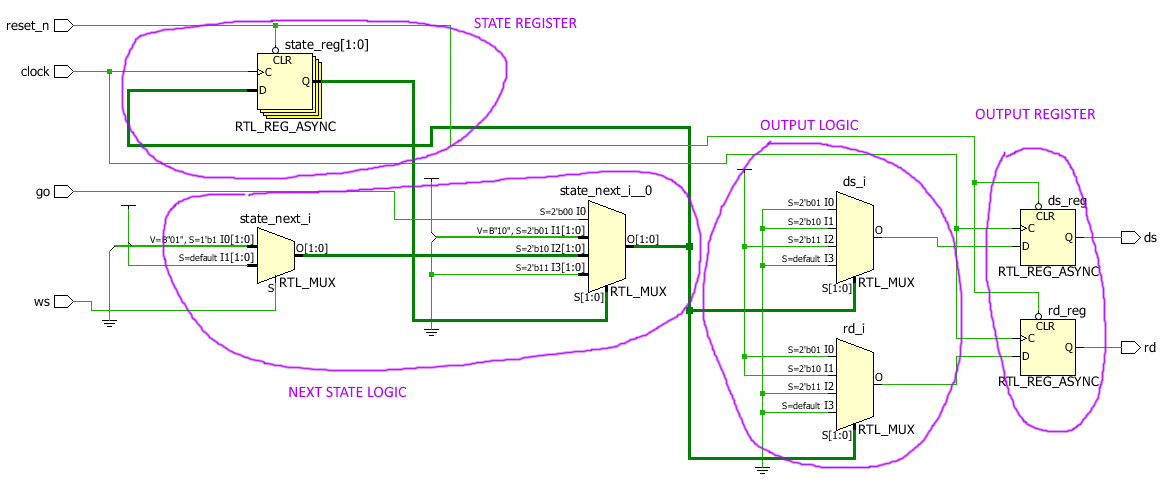


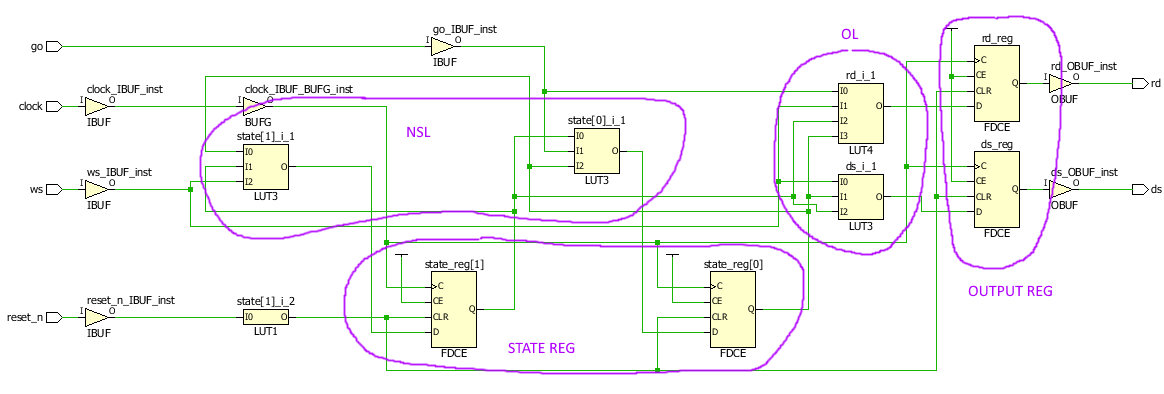


## Три always блока с регистрови изходи



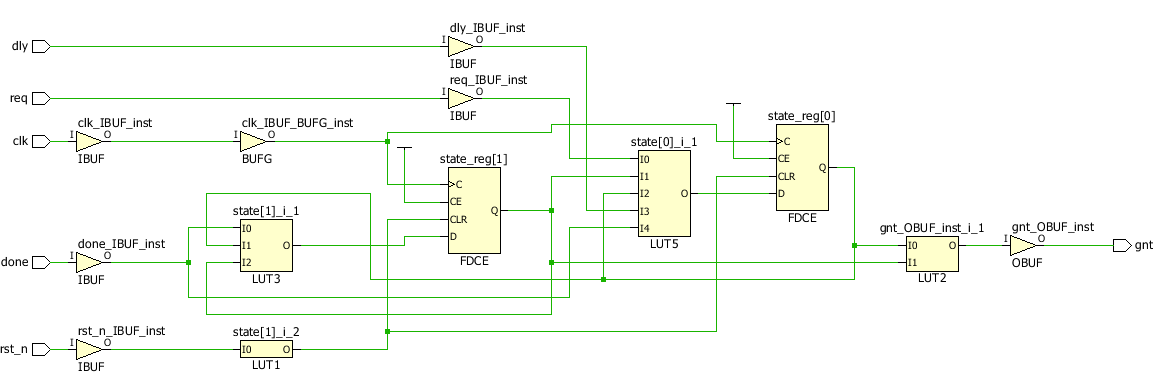
| module fsm1\_reg(  output logic rd, ds,  input go, ws, clock, reset\_n  );  typedef enum logic [1:0] {IDLE, READ, DELAY, DONE} state\_t;  state\_t state, state\_next;  // state register  always\_ff @(posedge clock, negedge reset\_n)  if (!reset\_n) state <= IDLE;  else state <= state\_next;  // next state logic  always\_comb begin  **state\_next = state;**  case (**state**)  IDLE:  if (go) state\_next = READ;  READ:  state\_next = DELAY;  DELAY:  if (ws) state\_next = READ;  else state\_next = DONE;  DONE:  state\_next = IDLE;  endcase  end  // output logic and registers  always\_ff @(posedge clock, negedge reset\_n)  if(!reset\_n) begin  ds <= '0;  rd <= '0;  end  else begin  rd <= '0;  ds <= '0;  case (**state\_next**)  READ : rd <= '1;  DELAY : rd <= '1;  DONE : ds <= '1;  endcase  end  endmodule | **always\_ff** блок описва регистъра на състоянията.  **alyaws\_com** блок задава преходите между състоянията.  **always\_ff** блок описва логическите функции на изходните сигнали и регистри  case (**state\_next**) NB! |
| --- | --- |



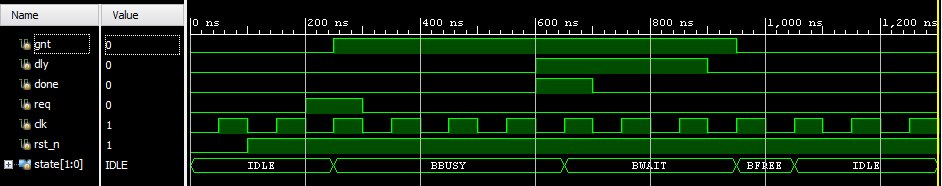


## Пример - fsm\_cc4\_2

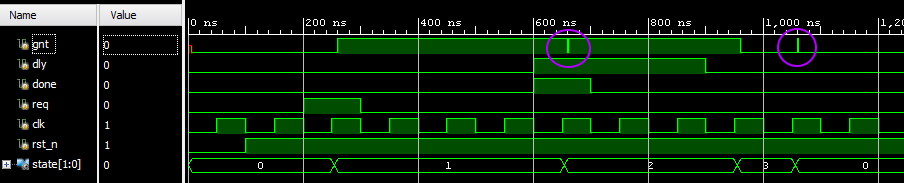
### Combinational outputs



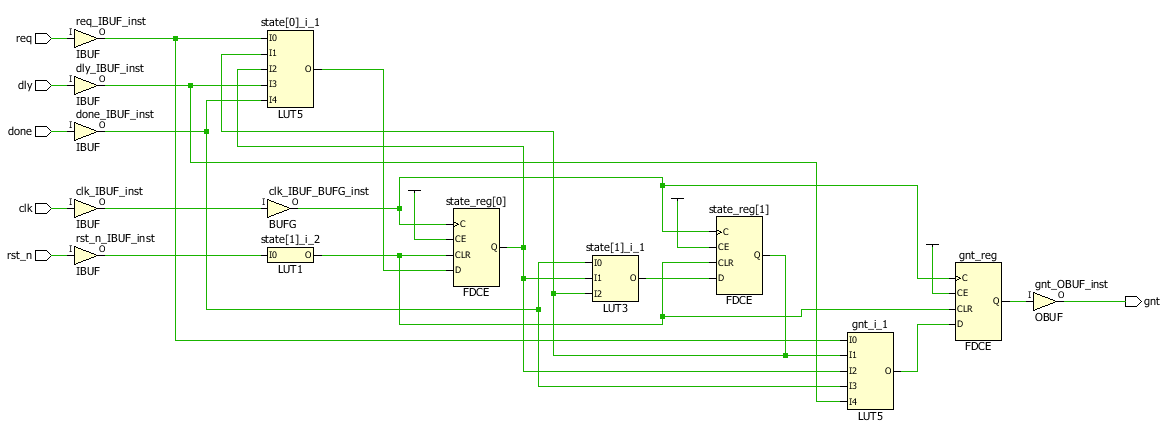
functional simulation



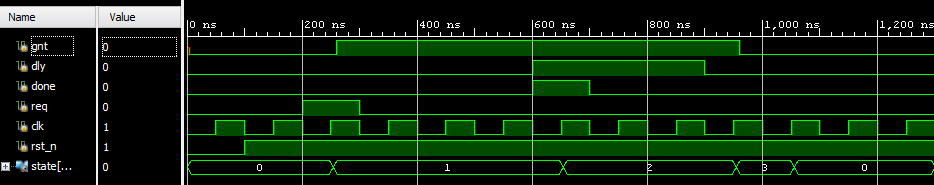
post routing timing simulation



### Registered outputs



post routing timing simulation

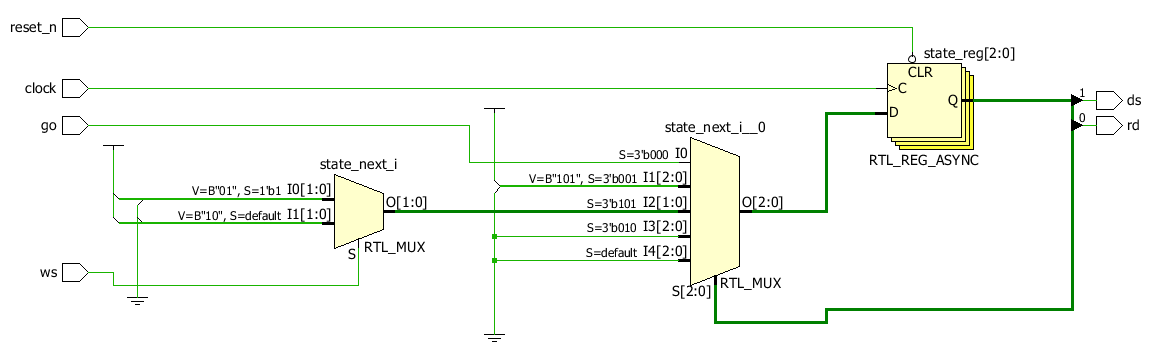


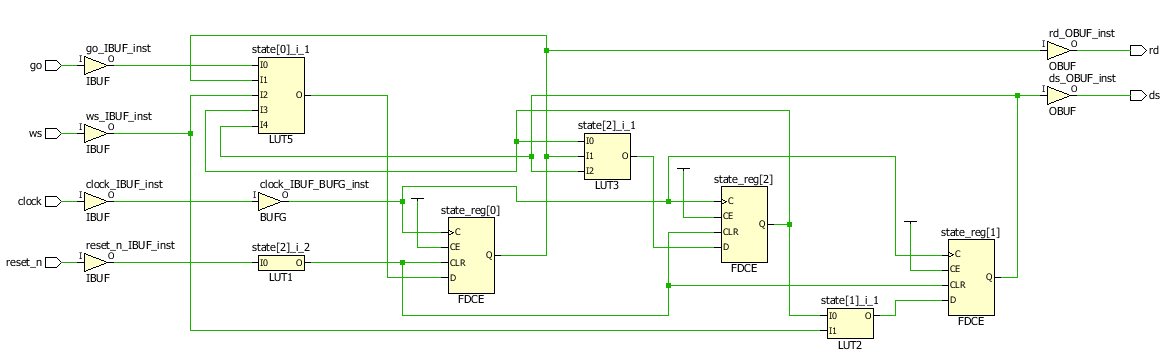
| module fsm\_cc4\_2(  output logic gnt,  input dly, done, req, clk, rst\_n);  enum logic [1:0] {  IDLE = 2'b00,  BBUSY = 2'b01,  BWAIT = 2'b10,  BFREE = 2'b11,  XX = 2'bxx} state, state\_next;  always\_ff @(posedge clk, negedge rst\_n)  if (!rst\_n) state <= IDLE;  else state <= state\_next;  always\_comb begin  state\_next = XX;  gnt = '0;  case (state)  IDLE :  if (req) state\_next = BBUSY;  else state\_next = IDLE;  BBUSY: begin  gnt = '1;  if (!done) state\_next = BBUSY;  else if (dly) state\_next = BWAIT;  else state\_next = BFREE;  end  BWAIT: begin  gnt = '1;  if (!dly) state\_next = BFREE;  else state\_next = BWAIT;  end  BFREE:  if (req) state\_next = BBUSY;  else state\_next = IDLE;  endcase  end  endmodule | module fsm\_cc4\_2\_**reg**(  output logic gnt,  input dly, done, req, clk, rst\_n);  enum logic [1:0] {  IDLE = 2'b00,  BBUSY = 2'b01,  BWAIT = 2'b10,  BFREE = 2'b11,  XX = 2'bxx} state, state\_next;  always\_ff @(posedge clk, negedge rst\_n)  if (!rst\_n) state <= IDLE;  else state <= state\_next;  always\_comb begin  state\_next = XX;  case (state)  IDLE :  if (req) state\_next = BBUSY;  else state\_next = IDLE;  BBUSY:  if (!done) state\_next = BBUSY;  else if (dly) state\_next = BWAIT;  else state\_next = BFREE;  BWAIT:  if (!dly) state\_next = BFREE;  else state\_next = BWAIT;  BFREE:  if (req) state\_next = BBUSY;  else state\_next = IDLE;  endcase  end  always\_ff @(posedge clk, negedge rst\_n)  if (!rst\_n) gnt <= '0;  else begin  case (state\_next)  IDLE, BFREE: gnt <= '0;  BBUSY, BWAIT: gnt <= '1;  endcase  end  endmodule |
| --- | --- |

## Включване на изходите в кодирането на състоянията



| module fsm1\_output\_encoding(  output logic rd, ds,  input go, ws, clock, reset\_n);  // state signals  enum logic [2:0]{  IDLE = 3'b0\_00,  READ = 3'b0\_01,  DLY = 3'b1\_01,  DONE = 3'b0\_10,  XX = 3'b0\_xx} state, state\_next;  assign {ds,rd} = state[1:0];  // state register  always\_ff @(posedge clock, negedge reset\_n)  if (!reset\_n) state <= IDLE;  else state <= state\_next;    // next state logic  always\_comb begin  state\_next = XX;  case (state)  IDLE:  if (go) state\_next = READ;  READ: state\_next = DLY;  DLY:  if (ws) state\_next = READ;  else state\_next = DONE;  DONE: state\_next = IDLE;  endcase  end  endmodule |  |
| --- | --- |





## Литература

1. Clifford E. Cummings, “Synthesizable Finite State Machine Design Techniques Using the New SystemVerilog 3.0 Enhancements”, SNUG Proceedings, San Jose 2003
2. Clifford E. Cummings, "State Machine Coding Styles for Synthesis," SNUG Proceedings, March 1998. Also available at www.sunburst-design.com/papers
3. Clifford E. Cummings, “Coding And Scripting Techniques For FSM Designs With Synthesis-Optimized, Glitch-Free Outputs,” SNUG Proceedings, September 2000. Also available at [www.sunburst-design.com/papers](http://www.sunburst-design.com/papers)
4. <https://en.wikipedia.org/wiki/Algorithmic_state_machine>