



```

// FSM by Mealy
module mealy_1101 ( y, x, clk, reset );
output y;
input  x;
input  clk;
input  reset;

reg    y;

parameter    // state identifiers
start  = 2'b00,
id1    = 2'b01,
id11   = 2'b11,
id110  = 2'b10;

reg [1:0] E1;// current state variables
reg [1:0] E2;// next state logic output

// next state logic
always @( x or E1 )
begin
y = `notfound;
case ( E1 )
start:
if ( x )
E2 = id1;
else
E2 = start;
id1:
if ( x )
E2 = id11;
else
E2 = start;
id11:
if ( x )
E2 = id11;
else
E2 = id110;
id110:
if ( x )
begin
E2 = id1;
y = `found;
end
else
begin
E2 = start;
y = `notfound;
end
default: // undefined state
E2 = 2'bxx;
endcase
end // always at signal or state changing

```

```
endmodule // mealy_1101
```

- ```
// -----
// --- Moore FSM
// -----
```

Moore FSM Diagram

```
graph LR
 start([start]) -- 1 --> id1([id1])
 id1 -- 1 --> id11([id11])
 id11 -- 0 --> id110([id110])
 id110 -- 1 --> id1101([id1101])
 id1101 -- 0 --> start
 id1101 -- 1 --> id1101
```

The diagram shows the following transitions:

- start** to **[id1]** on input **1**.
- [id1]** to **[id11]** on input **1**.
- [id11]** to **[id110]** on input **0**.
- [id110]** to **[id1101]** on input **1**.
- [id1101]** to **start** on input **0** (loop back to start).
- [id1101]** to **[id1101]** on input **1** (self-loop).

The final state is **[id1101]**, indicated by a double line.

```
// constant definition
`define found 1
`define notfound 0
```

```

// FSM by Moore
module moore_1101 (y, x, clk, reset);
output y;
input x;
input clk;
input reset;

reg y;

parameter // state identifiers
start = 3'b000,
id1 = 3'b001,
id11 = 3'b011,
id110 = 3'b010,
id1101 = 3'b110; // signal found

reg [2:0] E1;// current state variables
reg [2:0] E2;// next state logic output

// next state logic
always @(x or E1)
begin
 case(E1)
 start:
 if (x)
 E2 = id1;
 else
 E2 = start;
 id1:
 if (x)
 E2 = id11;
 else
 E2 = start;
 id11:
 if (x)
 E2 = id11;
 else
 E2 = id110;
 id110:
 if (x)
 E2 = id1101;
 else
 E2 = start;
 id1101:
 if (x)
 E2 = id11;
 else
 E2 = start;
 default: // undefined statee
 E2 = 3'bxxx;
 endcase
end // always at signal or state changing

```

```

// state variables
always @(posedge clk or negedge reset)
begin
 if (reset)
 E1 = E2; // updates current state
 else
 E1 = 0; // reset
 end // always at signal changing

// output logic
always @(E1)
begin
 y = E1[2]; // first bit of state value (MOORE indicator)
end // always at state changing

endmodule // moore_1101

```

- 03.) Projetar um circuito em Logisim para realizar a descrição em Verilog de um módulo para testar as máquinas de estados finitos segundo as abordagens de Mealy e de Moore. O nome do arquivo deverá ser Exemplo\_1101.v, e poderá seguir o modelo descrito abaixo.

```

// -----
// --- Mealy-Moore FSM
// -----
//

`include "mealy_1101.v"
`include "moore_1101.v"

module Exemplo1101;
 reg clk, reset, x;
 wire m1, m2;

 mealy_1101 mealy1 (m1, x, clk, reset);
 moore_1101 moore1 (m2, x, clk, reset);

 initial
 begin
 $display ("Time X Mealy Moore");

// initial values
 clk = 1;
 reset = 0;
 x = 0;
 end

```

```

// input signal changing
#5 reset = 1;
#10 x = 1;
#10 x = 0;
#10 x = 1;
#20 x = 0;
#10 x = 1;
#10 x = 1;
#10 x = 0;
#10 x = 1;

#30 $finish;
end // initial

always
#5 clk = ~clk;

always @(posedge clk)
begin
 $display ("%4d %4b %4b %5b", $time, x, m1, m2);
end // always at positive edge clocking changing

endmodule // Exemplo_1101

```

- 04.) Projetar um circuito em Logisim para realizar a  
 a descrição em Verilog de um módulo  
 para implementar uma máquina de estados finitos (FSM),  
 capaz de reconhecer apenas a primeira sequência  
 (0101) que aparecer.  
 uma sequência (0101) sem interseção  
 (0101010 não deverá ser reconhecida duas vezes).  
 O nome do arquivo deverá ser Exemplo\_1102.v.  
 Incluir previsão de testes e verificação do circuito pelo Logisim.
- 05.) Projetar um circuito em Logisim para realizar a  
 a descrição em Verilog de um módulo  
 para implementar uma máquina de estados finitos,  
 segundo a abordagem de Mealy, para reconhecer  
 uma sequência (0101) sem interseção  
 (01010101 deverá ser reconhecida apenas duas vezes).  
 O nome do arquivo deverá ser Exemplo\_1103.v.  
 Incluir previsão de testes e verificação do circuito pelo Logisim.
- 06.) Projetar um circuito em Logisim para realizar a  
 a descrição em Verilog de um módulo  
 segundo a abordagem de Moore, para reconhecer  
 uma sequência (0101) sem interseção  
 (010101010 deverá ser reconhecida três vezes).  
 O nome do arquivo deverá ser Exemplo\_1104.v.  
 Incluir previsão de testes e verificação do circuito pelo Logisim.

Extra

- 07.) Projetar um circuito em Logisim para realizar a descrição em Verilog de um módulo para implementar uma máquina de estados finitos, capaz de reconhecer uma sequência de quatro dígitos binários que termine com três valores iguais a 000 (x000, por exemplo). O nome do arquivo deverá ser Exemplo\_1105.v. Incluir previsão de testes e verificação do circuito pelo Logisim.
- 08.) Projetar um circuito em Logisim para realizar a descrição em Verilog de um módulo para implementar uma máquina de estados finitos, capaz de reconhecer uma sequência de três dígitos binários alternados (010 ou 101). O nome do arquivo deverá ser Exemplo\_1106.v. Incluir previsão de testes e verificação do circuito pelo Logisim.

Instruções para ver as cartas de tempo no GTKWave:

01.) Abrir o módulo de visualização (GTKWave)

02.) Selecionar a pasta de trabalho:

File

Open

Exemplo\_1101 (.vcd) (por exemplo)

03.) Selecionar os sinais desejados:

clk (sinal a ser visto)

clock (outro sinal a ser visto)

(selecionar, arrastar e soltar na coluna à direita)

Modelo em Logisim para um detector de sequência 1101

Sequence detector

