طراحی کامپیوتری سیستم‌های دیجیتال

جلسه ششم



تخصیص روالی

تخصیص روالی: تخصیصی از یک مقدار منطقی به یک متغییر درون یک گزاره initial یا always می باشد.

دو نوع تخصیص روالی وجود دارد:

- Blocking که با علامت (=) مشخص می شود.

```
B = A
C = B + 1
```

- NON-Blocking که از عملگر (<=) استفاده می کند.

```
B <= A
C <= B + 1
```

Blocking Assignments

گزاره های تخصیص بلوکه ساز:

به ترتیبی که آنها در یک بلوک از گزاره ها فهرست شده اند اجرا می شوند.

```
B = A
C = B + 1
```

C حاوی مقدار A+1 خواهد بود.

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial begin
  x=0; y=1; z=1;
  count=0;
  reg_a= 16'b0; reg_b = reg_a;
  #15 reg a[2] = 1'b1;
  #10 reg_b[15:13] = {x, y, z};
  count = count + 1;
end
```

All executed at time 0

executed at time 15

All executed at time 25

Non-Blocking Assignments

گزاره های تخصیص غیربلوکه ساز:

ابتدا عبارات سمت راست ارزیابی شده ودر مکانی موقتی ذخیره می شوند ، بعد از اینکه تمام عبارات بلوک ارزیابی شد تخصیص به اهداف سمت چپ به عمل می آید.

B <= A
C <= B + 1

C حاوی مقدار اولیه B به علاوه ی 1 خواهد بود.

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial begin
  x=0; y=1; z=1;
  count=0;
  reg_a= 16'b0; reg_b = reg_a;
  reg_a[2] <= #15 1'b1;
  reg_b[15:13] <= #10 {x, y, z};
  count <= count + 1;
end
```

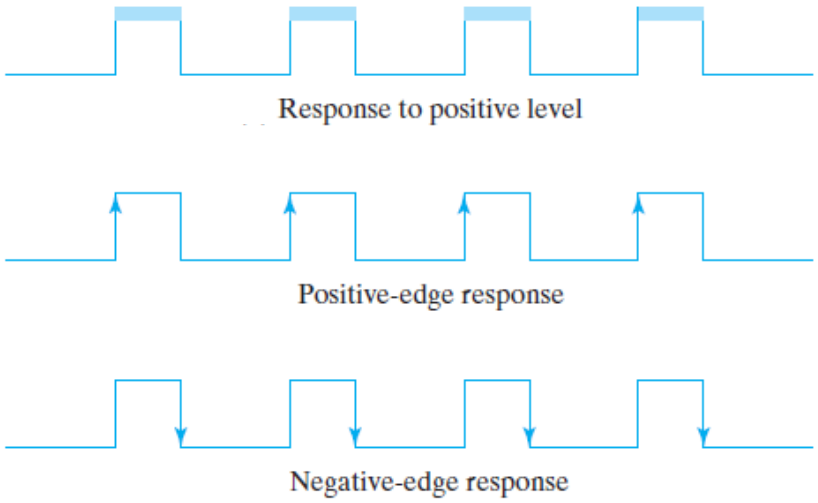
All executed at time 0

Scheduled to run at time 15

Scheduled to run at time 10

کاربرد

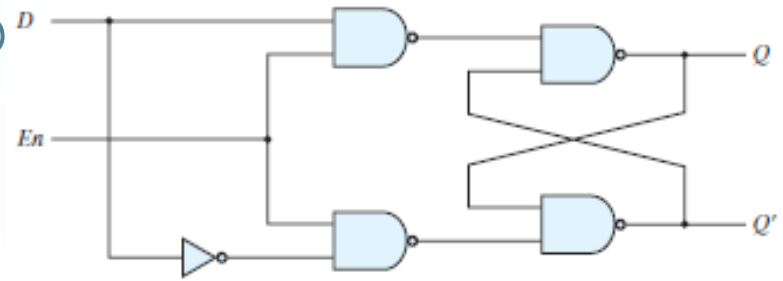
- از تخصیص های بلوکه ساز در مواقعی که مرتب سازی ترتیبی ضروررری است و همچنین در رفتارهای چرخه ای حساس به سطح و منطق ترکیبی استفاده می شود.
- از تخصیص های غیر بلوکه ساز برای اجرای همزمان (انتقال های ثبات همزمان) و مدل سازی نگهدار (Latch) استفاده می شود.
همچنین برای مدل سازی رفتارهای حساس به لبه.



D Latch

لچ ها: عناصر ذخیره سازی که با سطوح سیگنال کار می کنند.

"عناصرحساس به سطح"



En	D	Next state of Q
0	X	No change
1	0	Q = 0; reset state
1	1	Q = 1; set state

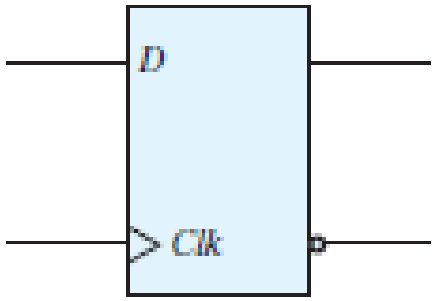
```
module D_latch (Q, D, enable);
  output Q;
  input   D, enable;
  reg     Q;
  always @ (enable or D)
    if (enable) Q <= D;                    // Same as: if (enable == 1)
endmodule

// Alternative syntax (Verilog 2001, 2005)
module D_latch (output reg Q, input enable, D);
  always @ (enable, D)
    if (enable) Q <= D;                    // No action if enable not asserted
endmodule
```

D FLIP-FLOP

عناصر ذخیره سازی که با یک گذر پالس ساعت کنترل می شود.

"عناصر حساس به لبه"



(a) Positive-edge

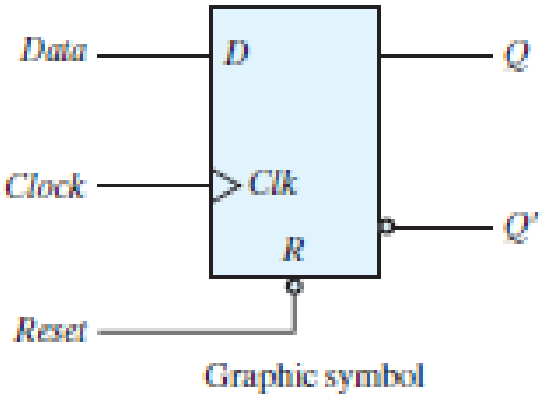
D Flip-Flop

D	Q(t + 1)	
0	0	Reset
1	1	Set

```
// D flip-flop without reset
module D_FF (Q, D, Clk);
  output Q;
  input  D, Clk;
  reg    Q;
  always @ (posedge Clk)
     Q <= D;
  endmodule
```

فلیپ فلاپ D

همراه با Reset



Graphic symbol

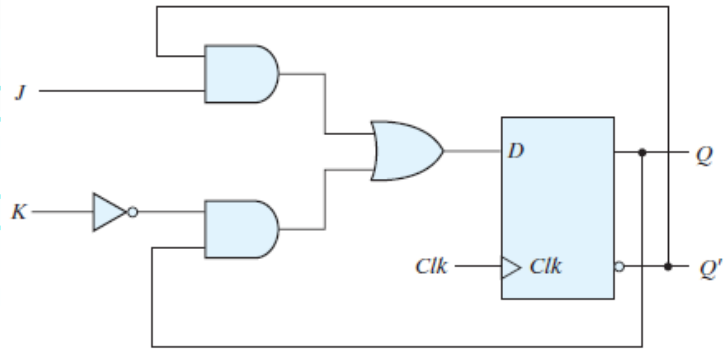
```
// D flip-flop with asynchronous reset (V2001, V2005)
module DFF (output reg Q, input D, Clk, rst);
  always @ (posedge Clk, negedge rst)
  if (!rst) Q <= 1'b0;          // Same as: if (rst == 0)
  else Q <= D;
endmodule
```

JK FLIP-FLOP

فلیپ فلاپ JK با استفاده از فلیپ فلاپ D

```
// JK flip-flop from D flip-flop and gates (V2001, 2005)
module JKFF (output reg Q, input J, K, Clk, rst);
  wire JK;
  assign JK = (J & ~Q) | (~K & Q);
// Instantiate D flip-flop
  DFF JK1 (Q, JK, Clk, rst);
endmodule

// D flip-flop (V2001, V2005)
module DFF (output reg Q, input D, Clk, rst);
  always @ (posedge Clk, negedge rst)
    if (!rst) Q <= 1'b0;
    else Q <= D;
endmodule
```

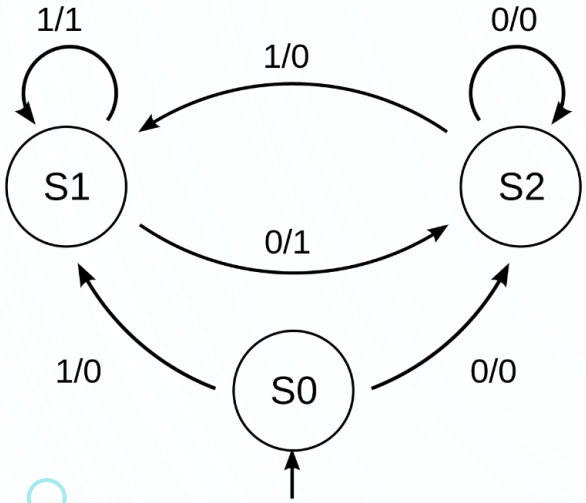


JK Flip-Flop

J	K	Q(t + 1)	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

ماشین متناهی-مدل میلی

یک نوع از ماشین های حالت های متناهی است که خروجی آن به حالت کنونی و ورودی کنونی وابسته است.



دیاگرام حالت

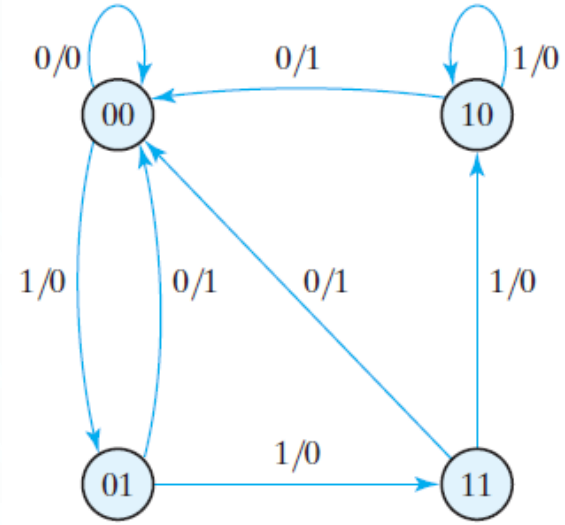
Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
S0	S2	S1	0	0
S1	S2	S1	1	1
S2	S2	S1	0	0

جدول حالت

آشکارساز صفر (میلی)

تشخیص صفر در ورودی در میان رشته ای از یک ها

Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0



آشکار ساز صفر(میلی)

مدلسازی رفتاری

```
module Mealy_Zero_Detector (
  output reg y_out,
  input   x_in, clock, reset
);
  reg [1: 0]          state, next_state;
  parameter           S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
  always @ (posedge clock, negedge reset)    Verilog 2001, 2005 syntax
    if (reset == 0) state <= S0;
    else state <= next_state;

  always @ (state, x_in)                          // Form the next state
    case (state)
       S0:        if (x_in)  next_state = S1; else next_state = S0;
       S1:        if (x_in)  next_state = S3; else next_state = S0;
       S2:        if (~x_in)  next_state = S0; else next_state = S2;
       S3:        if (x_in)  next_state = S2; else next_state = S0;
    endcase

  always @ (state, x_in)                          // Form the Mealy output
    case (state)
       S0:          y_out = 0;
       S1, S2, S3: y_out = ~x_in;
    endcase
endmodule
```

آشکار ساز صفر (میلی)

میز تست

```
module t_Mealy_Zero_Detector;
 wire     t_y_out;
 reg      t_x_in, t_clock, t_reset;

Mealy_Zero_Detector M0 (t_y_out, t_x_in, t_clock, t_reset);
initial #200 $finish;
initial begin t_clock = 0; forever #5 t_clock = ~t_clock; end

initial fork
      t_reset = 0;
 #2 t_reset = 1;
 #87 t_reset = 0;
 #89 t  reset = 1;
 #10 t_x_in = 1;
 #30 t_x_in = 0;
 #40 t_x_in = 1;
 #50 t_x_in = 0;
 #52 t_x_in = 1;
 #54 t_x_in = 0;
 #70 t_x_in = 1;
 #80 t_x_in = 1;
 #70 t_x_in = 0;
 #90 t_x_in = 1;
 #100 t_x_in = 0;
 #120 t_x_in = 1;
 #160 t_x_in = 0;
 #170 t_x_in = 1;
 join
endmodule
```

fork….join

اجرای موازی بجای اجرای ترتیبی را سبب می شود.

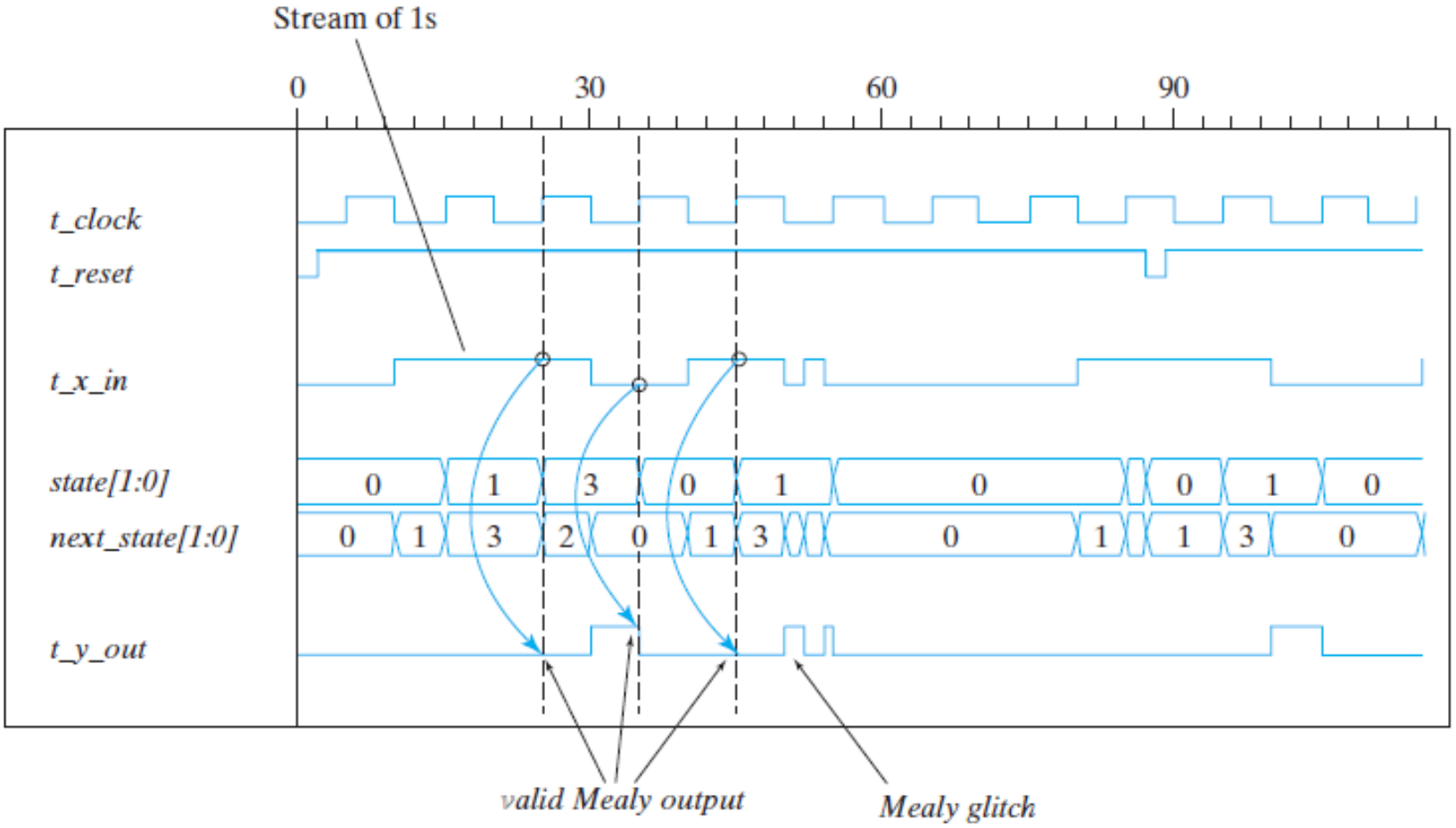
```
initial begin:begin_end
fork
begin:fj_b1
  #2 $display("@%0t:fj_b1",$time);
end
begin: fj_b2
  #1 $display("@%0t:fj_b2",$time);
end
#3 display("@%0t:fj_3",$time);
join
#1 $display("@%0t:begin_end",$time);
#5;
end
```

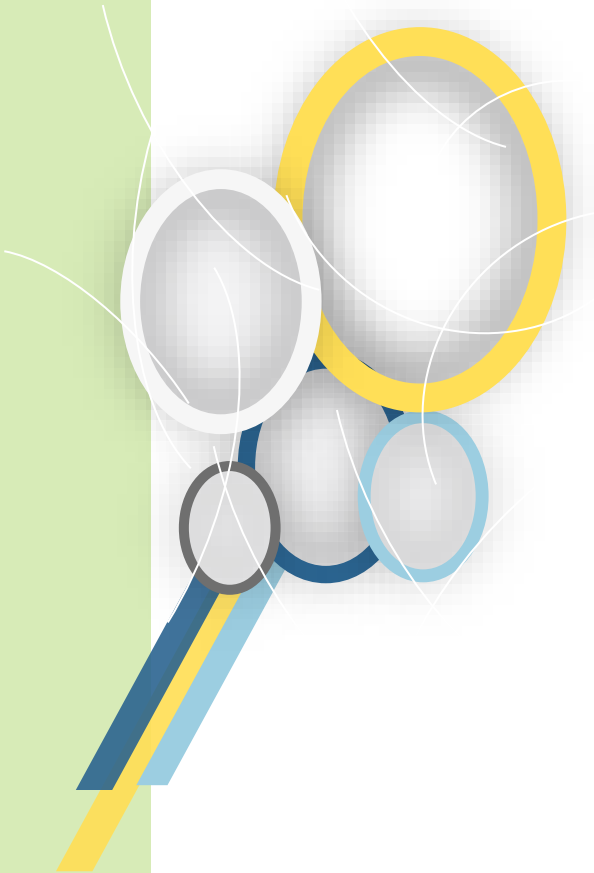
جواب

@1:fj_b2
@2:fj_b1
@3:fj_3
@4:begin_end

آشکار ساز صفر

سیگنال های ورودی و خروجی





خسته نباشید