@*
@(*)

Example 7-17 Use of @ Operator*

```
//Combination logic block using the or operator
//Cumbersome to write and it is easy to miss one input to the block
always @(a or b or c or d or e or f or g or h or p or m)

begin
out1 = a ? b+c : d+e;
out2 = f ? g+h : p+m;
end

//Instead of the above method, use @(*) symbol
//Alternately, the @* symbol can be used
//All input variables are automatically included in the
//sensitivity list.
always @(*)
begin
out1 = a ? b+c : d+e;
out2 = f ? g+h : p+m;
end
```

Example 7-20 Case Statement with x and z

```
module demultiplexer1_to_4 (out0, out1, out2, out3, in, s1, s0);

// Port declarations from the I/O diagram
output out0, out1, out2, out3;
reg  out0,  out1,  out2,  out3;
input in;
input s1, s0;

always @(s1 or s0 or in)
case ({s1, s0}) //Switch based on control signals
    2'b00 : begin  out0 = in;  out1 = 1'bz;  out2 = 1'bz; out3 = 1'bz; end
    2'b01 : begin  out0 = 1'bz;  out1 = in;  out2 = 1'bz; out3 = 1'bz; end
    2'b10 : begin  out0 = 1'bz;  out1 = 1'bz;  out2 = in; out3 = 1'bz; end
    2'b11 : begin  out0 = 1'bz;  out1 = 1'bz;  out2 = 1'bz; out3 = in; end

    //Account for unknown signals on select. If any select signal is x
    //then outputs are x. If any select signal is z, outputs are z.
    //If one is x and the other is z, x gets higher priority.
    2'bx0, 2'bx1, 2'bxz, 2'bxx, 2'b0x, 2'b1x, 2'bzx :
         begin
              out0 = 1'bx;  out1 = 1'bx;  out2 = 1'bx;  out3 = 1'bx;
         end
    2'bz0, 2'bz1, 2'bzz, 2'b0z, 2'b1z :
         begin
              out0 = 1'bz;  out1 = 1'bz;  out2 = 1'bz;  out3 = 1'bz;
         end
    default: $display("Unspecified control signals");
endcase

endmodule
```

Example 7-26 Sequential Blocks

```
//Illustration 1: Sequential block without delay
reg x, y;
reg [1:0] z, w;

initial
begin
        x = 1'b0;
        y = 1'b1;
        z = {x, y};
        w = {y, x};
end

//Illustration 2: Sequential blocks with delay.
reg x, y;
reg [1:0] z, w;

initial
begin
        x = 1'b0; //completes at simulation time 0
        #5 y = 1'b1; //completes at simulation time 5
        #10 z = {x, y}; //completes at simulation time 15
        #20 w = {y, x}; //completes at simulation time 35
end
```

Example 7-27 Parallel Blocks

```
//Example 1: Parallel blocks with delay.
reg x, y;
reg [1:0] z, w;

initial
fork
        x = 1'b0; //completes at simulation time 0
        #5 y = 1'b1; //completes at simulation time 5
        #10 z = {x, y}; //completes at simulation time 10
        #20 w = {y, x}; //completes at simulation time 20
join
```

Example 7-35 *Behavioral 4-to-1 Multiplexer*

```
// 4-to-1 multiplexer. Port list is taken exactly from
// the I/O diagram.
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);

// Port declarations from the I/O diagram
output out;
input i0, i1, i2, i3;
input s1, s0;
//output declared as register
reg out;

//recompute the signal out if any input signal changes.
//All input signals that cause a recomputation of out to
//occur must go into the always @(...)  sensitivity list.
always @(s1 or s0 or i0 or i1 or i2 or i3)
begin
  case ({s1, s0})
  2'b00: out = i0;
  2'b01: out = i1;
  2'b10: out = i2;
  2'b11: out = i3;
  default: out = 1'bx;
  endcase
end


endmodule
```

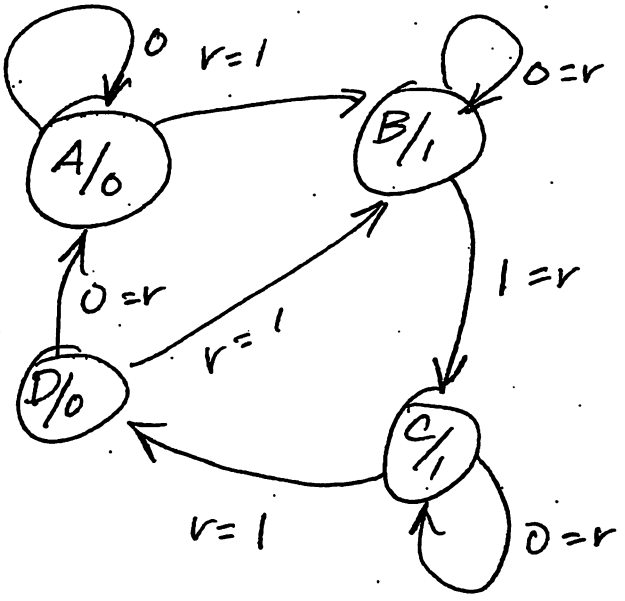
Example 7-36 *Behavioral 4-bit Counter Description*

```
//4-bit Binary  counter
module counter(Q , clock, clear);

// I/O ports
output [3:0] Q;
input clock, clear;
//output defined as register
reg [3:0] Q;

always @( posedge clear  or negedge clock)
begin
  if (clear)
     Q <= 4'd0;  //Nonblocking assignments are recommended
                 //for creating sequential logic such as flipflops
  else
     Q <= Q + 1;// Modulo 16 is not necessary because Q is a
                // 4-bit value and wraps around.
end

endmodule
```

Note:
Initial value
for regs is "x"

Let's talk about modeling FSMs

Moore



reset

```
module moore_fsm (r,clock,Z);
input r, clock;
output Z;
reg Z;

//make state assignment (full encoded)
parameter A=0,B=1,C=2,D=3;
reg [1:0] state;

always @(posedge clock)  // update state
     case(state)
          A: if(r) state=B;
          B: if(r) state=C;
          C: if(r) state=D;
          D: if(r) state=B else state=A;
     endcase

always@(state)  // update output
   case(state)
        A,D: Z=0;
        B,C: Z=1;
  endcase

endmodule
```

reset

if(!reset)
  state=A;
else

Mealy

	next r=0	next r=1	output r=0	output r=1
A	A	D	0	1
B	B	A	1	0
C	C	B	0	1
D	C	B	0	0

```
Module Mealy_FSM (r, clock, reset, z);
 input r, clock, reset;
 output z;
  reg  z;
parameter A=1, B=2, C=3, D=4;
 reg [1:0] present, next;
 // sequential part
 always @(posedge clock or
                      negedge reset)
        if (! reset)
              present = A;
        else
              present = next;
```

```
always @ (present or r) // must check for both
begin: combinational_part
  case (present)
    A:  if (r)
        begin z=1; next=D; end
        else z=0;

    B:  if (r)
        begin z=0; next=A; end
        else z=1;

    C:  if (r)
        begin z=1; next=B; end
        else z=0;

    D.  if (r)
        begin z=0; next=B; end
        else
        begin z=0; next=C; end

  endcase
end // comb_part

endmodule
```

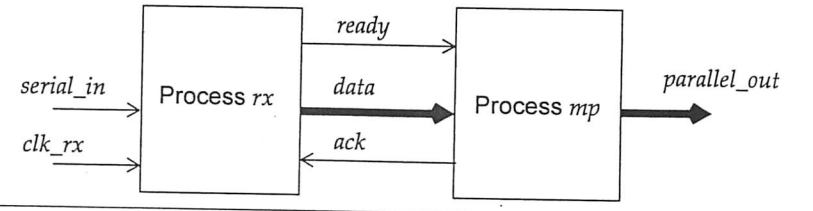


Figure 8-8 Two interacting processes.

```
`timescale 1ns/100ps
module interacting (serial_in, clk_rx, parallel_out);
  input serial_in, clk_rx;
  output reg [7:0] parallel_out;

  reg ready, ack;
  reg [0:7] data;

  `include "read_word.v" // Task read_word is defined in
                         // this file.
  always
    begin: RX
      read_word (serial_in, clk_rx, data);
      // The task read_word reads the serial data on every
      // clock cycle and converts to a parallel data in
      // signal data. It takes 10ns to do this.
      ready <= 1;
      wait (ack);
      ready <= 0;
      #40;
    end

  always
    begin: MP
      #25;
      parallel_out <= data;
      ack <= 1;
      #25 ack <= 0;
      wait (ready);
    end
endmodule
```