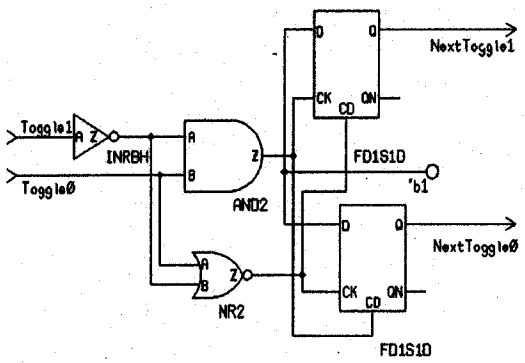
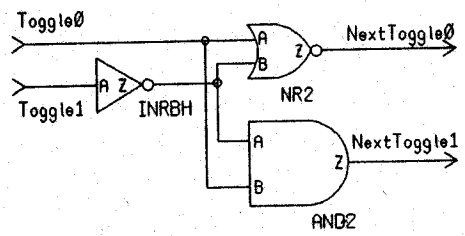
```
module NextStateLogic (NextToggle, Toggle);
  input [1:0] Toggle;
  output [1:0] NextToggle;
  reg [1:0] NextToggle;

  always @ (Toggle)
    case (Toggle)
      2'b01 : NextToggle = 2'b10;
      2'b10 : NextToggle = 2'b01;
    endcase

endmodule
```

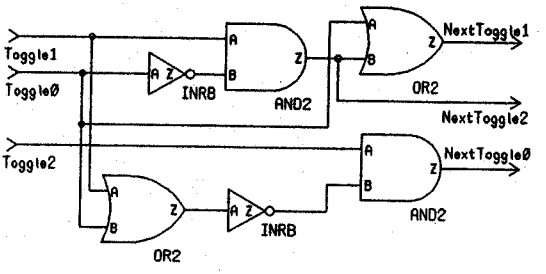


```
module NextStateLogicFullCase (NextToggle, Toggle);
  input [1:0] Toggle;
  output [1:0] NextToggle;
  reg [1:0] NextToggle;


    always @ (Toggle)
      case (Toggle)                    // synthesis full_case
        2'b01 : NextToggle = 2'b10;
        2'b10 : NextToggle = 2'b01;
      endcase
  endmodule
```



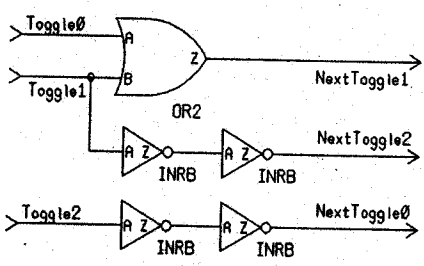
```
module PriorityLogic (NextToggle, Toggle);
  input [2:0] Toggle;
  output [2:0] NextToggle;
  reg [2:0] NextToggle;

  always @ (Toggle)
    casex (Toggle)
      3'bxx1 : NextToggle = 3'b010;
      3'bx1x : NextToggle = 3'b110;
      3'b1xx : NextToggle = 3'b001;
      default : NextToggle = 3'b000;
    endcase
endmodule
```

```
  if (Toggle[0] == 'b1)
    NextToggle = 3'b010;
  else if (Toggle[1] == 'b1)
    NextToggle = 3'b110;
  else if (Toggle[2] == 'b1)
    NextToggle = 3'b001;
  else
    NextToggle = 3'b000;
```

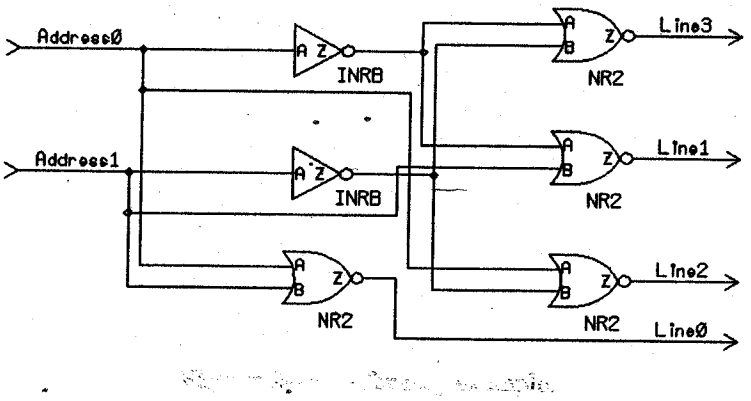


```
module ParallelCase (NextToggle, Toggle);
  input [2:0] Toggle;
  output [2:0] NextToggle;
  reg [2:0] NextToggle;

  always @ (Toggle)
    casex (Toggle)              // synthesis parallel_case
      3'bxx1 : NextToggle = 3'b010;
      3'bx1x : NextToggle = 3'b110;
      3'b1xx : NextToggle = 3'b001;
      default : NextToggle = 3'b000;
    endcase
endmodule
```

```
if (Toggle[0] == 'b1)
  NextToggle = 3'b010;

if (Toggle[1] == 'b1)
  NextToggle = 3'b110;

if (Toggle[2] == 'b1)
  NextToggle = 3'b001;

if ((Toggle[0] != 'b1) &&
    (Toggle[1] != 'b1) &&
    (Toggle[2] != 'b1))
  NextToggle = 3'b000;
```

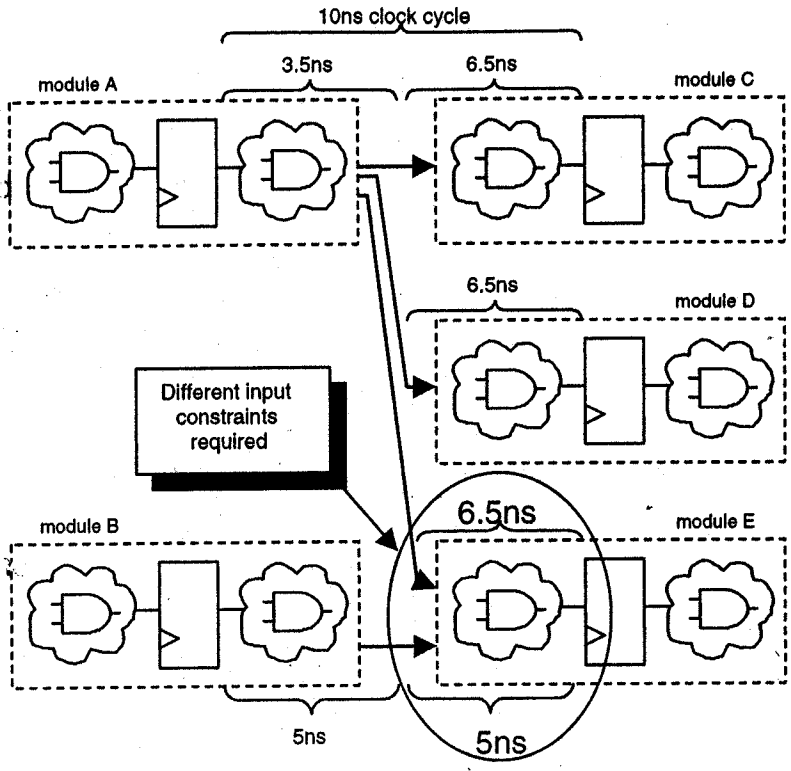


```
  output [3:0] Line;
  reg [3:0] Line;

  integer J;

  always @ (Address)
    for (J = 3; J >= 0; J = J - 1)
      if (Address == J)
        Line[J] = 1;
      else
        Line[J] = 0;
endmodule
```




When the for-loop is expanded, the following four `if` statements are obtained.

```
if (Address == 3) Line[3] = 1; else Line[3] = 0;
if (Address == 2) Line[2] = 1; else Line[2] = 0;
if (Address == 1) Line[1] = 1; else Line[1] = 0;
if (Address == 0) Line[0] = 1; else Line[0] = 0;
```




Constraining combinational outputs that drive combinational inputs