Digital Design with Verilog

Verilog - Synthesis

Lecture 22: Logic Synthesis with Verilog HDL – Part5

Objectives

- Loop Statements
- Modeling Flip Flops
- Use of Local Variables

Loop Statement

- There are four kinds of loops
  - While-loop
  - For-loop
  - Forever-loop
  - Repeat-loop
- The For-loop alone is typically supported for synthesis, with a restriction

Synthesis of For Loops

- Unrolling the For loop –All statements within the For loop are replicated, once for each value of the For-loop index.

- The restriction is that the loop bounds have to be constants –else synthesis is beyond any scope

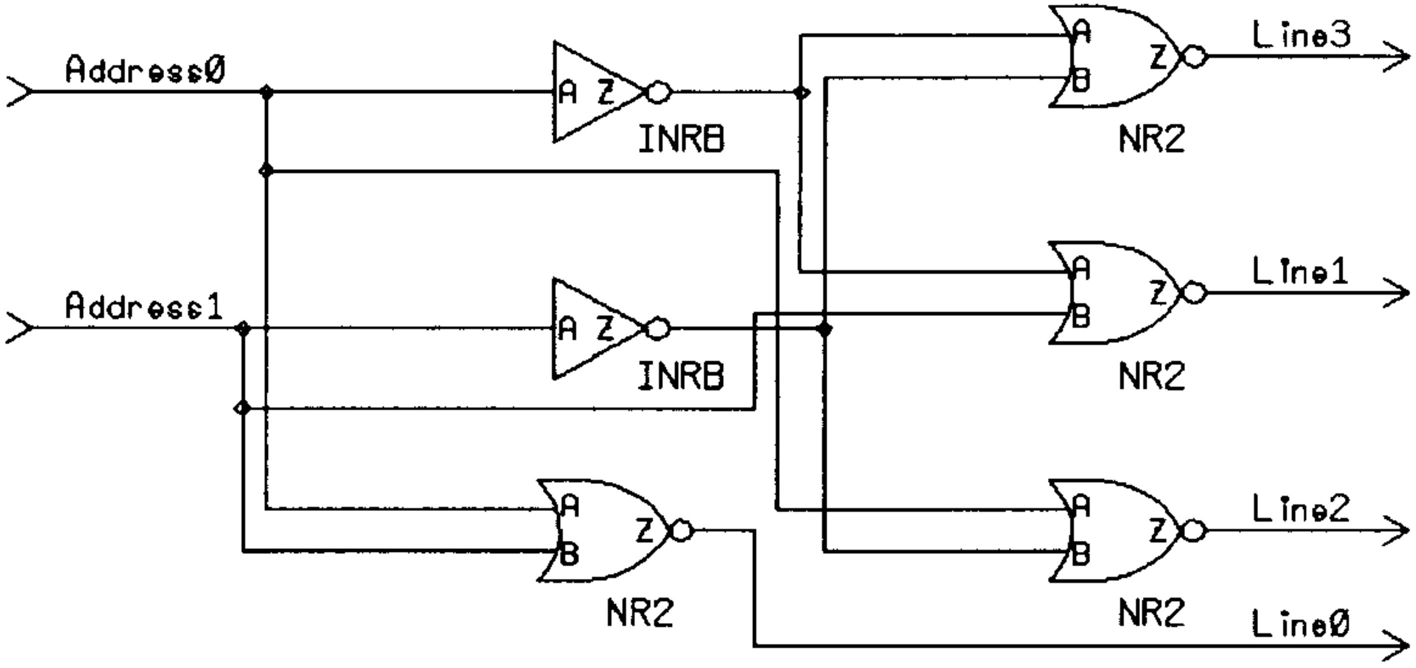
Synthesis of For Loops

```
module DeMultiplexer (Address, Line);
input [1:0] Address;
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
```

Loop Unrolling

- 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;

Synthesis of For Loops



A for-loop example

Modeling Flip-flops

- A variable that is assigned a value inside a special always construct gets modeled as a flip-flop.

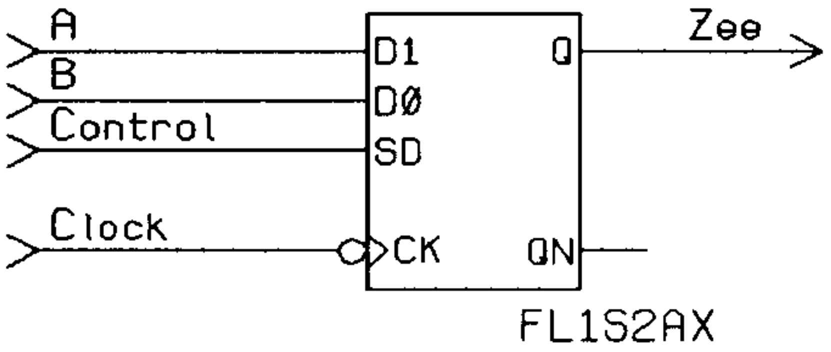
- The special always construct is:

  *always @(<clock_event>),*where a *<clock_event>* is *posedge<clock_name>* or *negedge<clock_name>.*

- This is called a *clocked always statement*.

Modeling Flip-flops

```
module PickOne (A, B, Clock, Control, Zee);
input A, B, Clock, Control;
output Zee;
reg Zee;

always @ (negedge Clock)
    if (Control)
        Zee <=A;
    else
        Zee <= B;
endmodule
```



Sequential logic synthesized using a special always statement.

Explanation

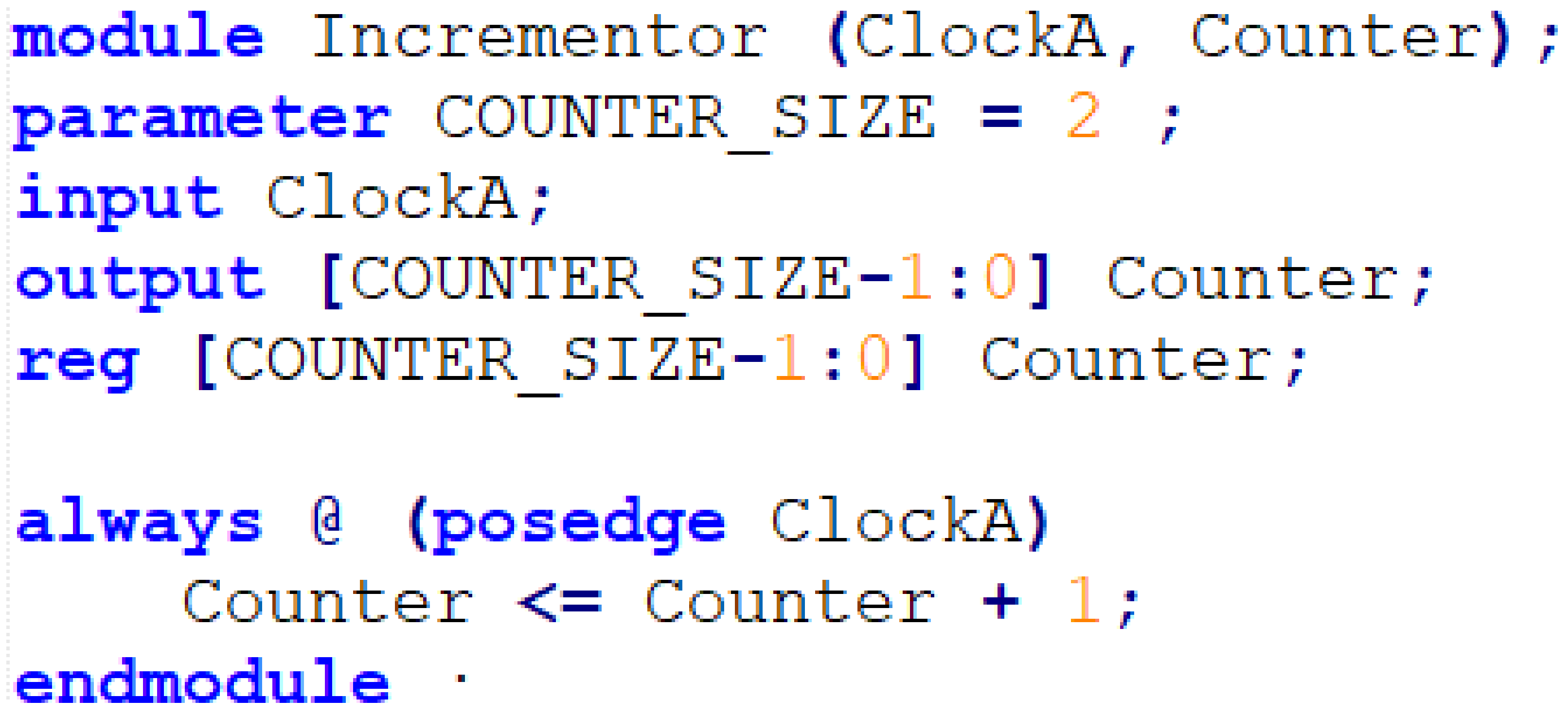
- On the *Control* Variable –D0 or D1 is selected and the logic is inherent in the flip-flop definition.
- The Clock is given to the clock input –the expression following *negedge* or *posedge* is given as the clock input.
- The variable *Zee* is modeled as a flip-flop

Non Blocking Assignment

- This is used in sequential logic to avoid any functional mismatch between the HDL code and the synthesized netlist, specifically for variables that are assigned in a clocked always statement and its value used outside.

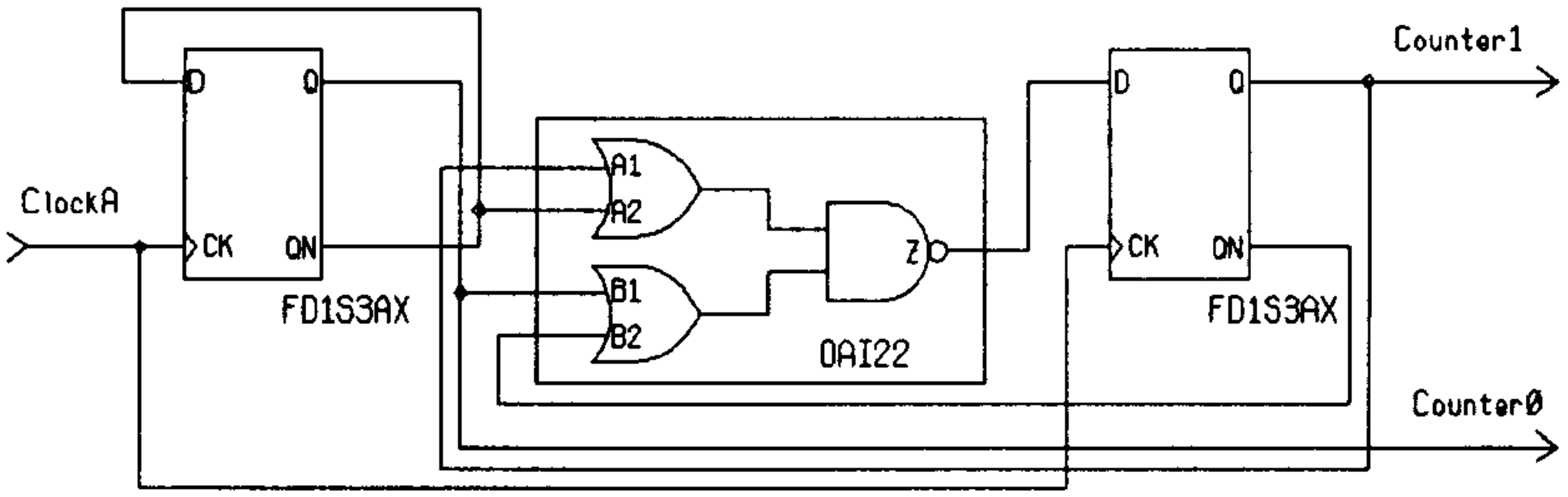
- Why? –Of all assignments, the non-blocking ones are the last to be executed.

Modeling Flip-flops



```
module Incrementor (ClockA, Counter);
parameter COUNTER_SIZE = 2 ;
input ClockA;
output [COUNTER_SIZE-1:0] Counter;
reg [COUNTER_SIZE-1:0] Counter;

always @ (posedge ClockA)
    Counter <= Counter + 1;
endmodule .
```

Modeling Flip-flops

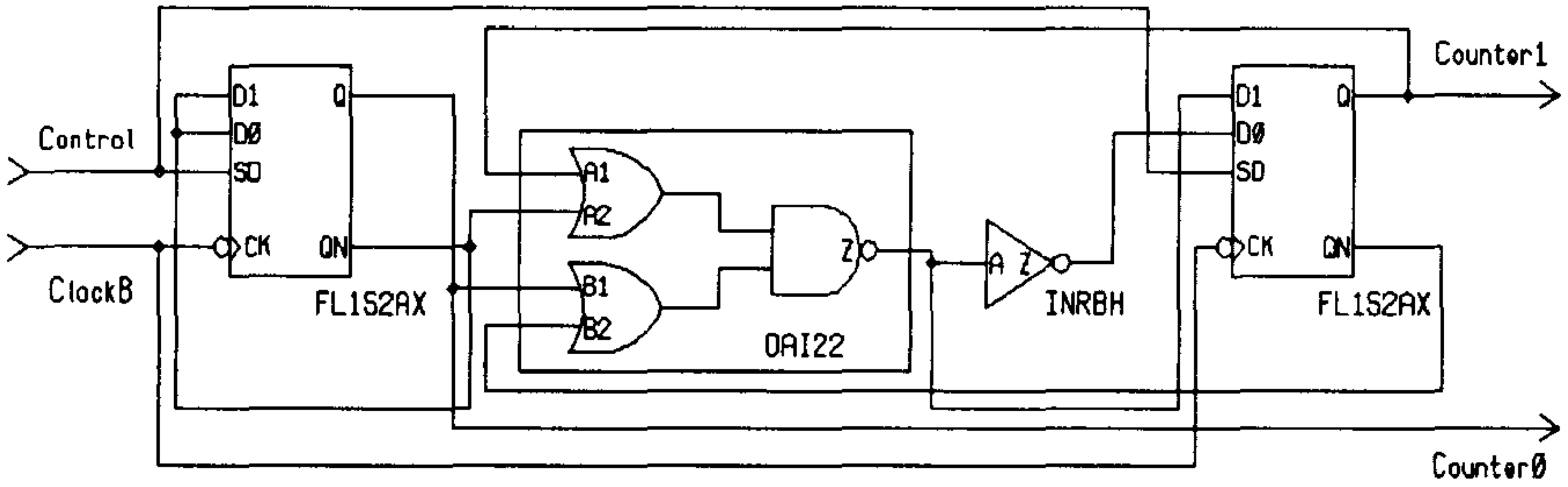


Modeling flip-flops

Modeling Flip-flops

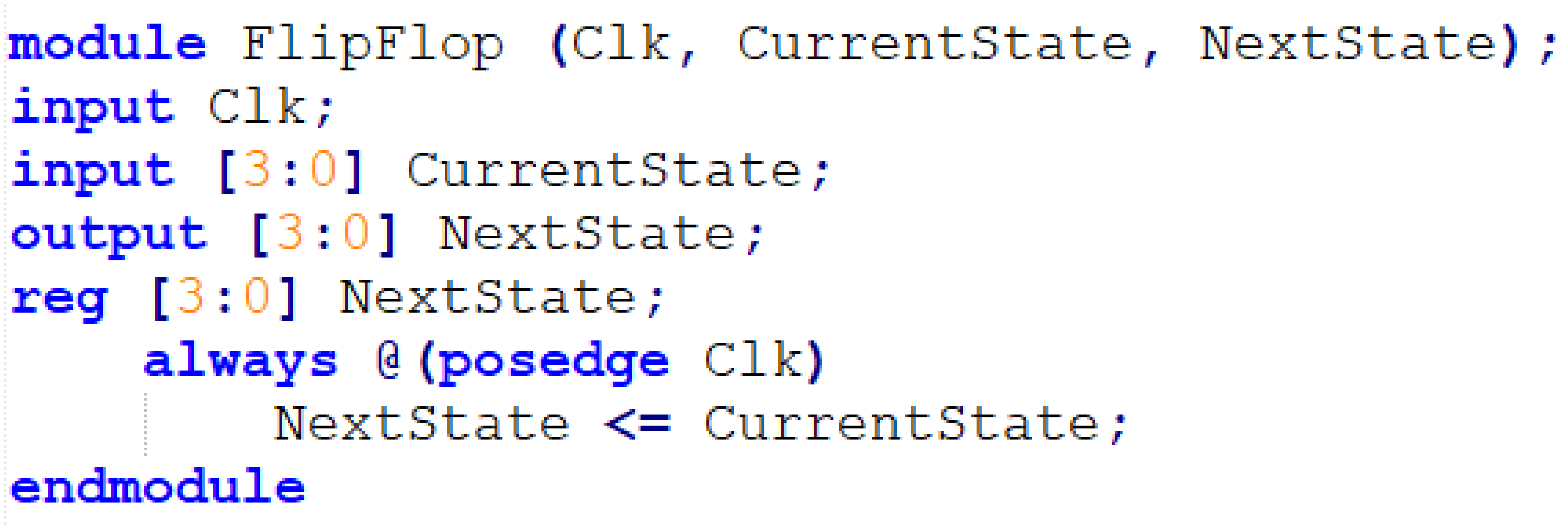
```
module UpDownCounter (Control, ClockB, Counter);
input Control, ClockB;
output [1:0] Counter;
reg [1:0] Counter;

always @ ( negedge ClockB)
    if (Control)
        Counter <= Counter+ 1;
    else
        Counter <= Counter- 1;
endmodule
```

Modeling Flip-flops



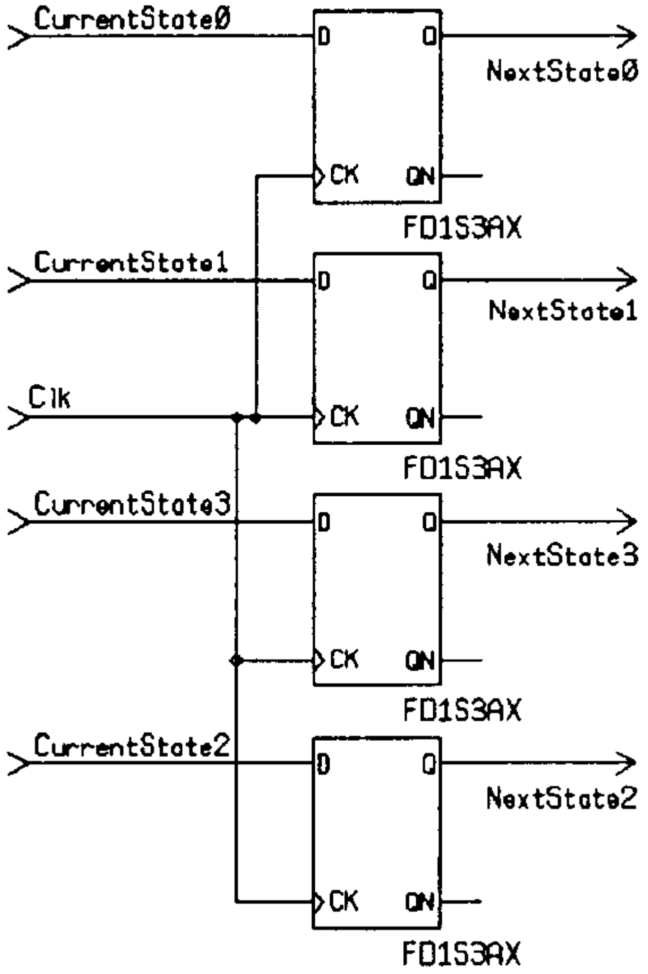
Falling-edge-triggered flip-flops inferred.

Modeling Flip-flops



```
module FlipFlop (Clk, CurrentState, NextState);
input Clk;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;
    always @(posedge Clk)
        NextState <= CurrentState;
endmodule
```

Modeling Flip-flops



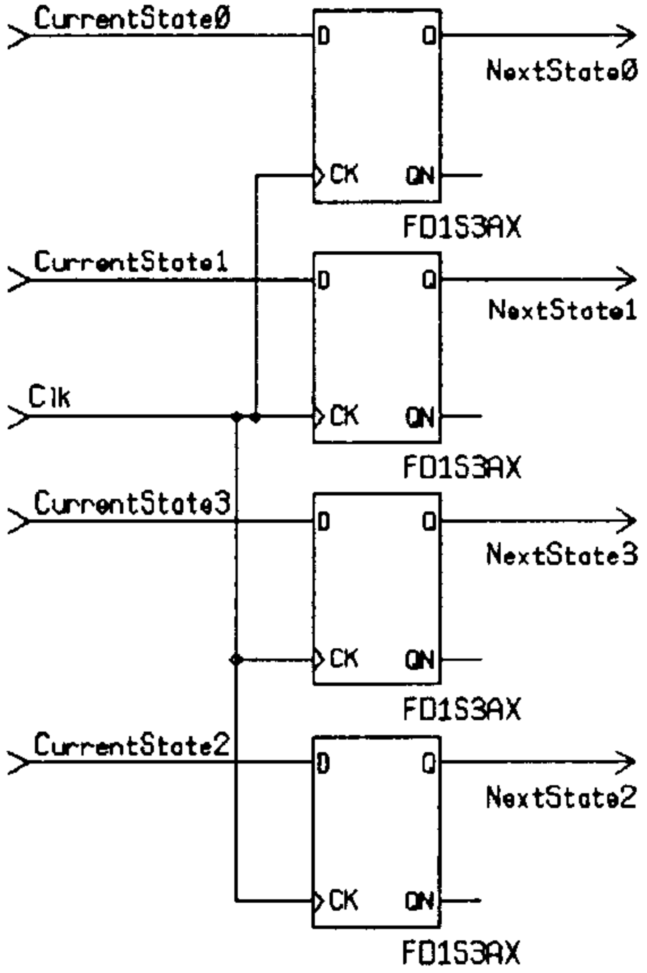
Flip-flops inferred from a variable assigned under clock control

Local Use of Variables

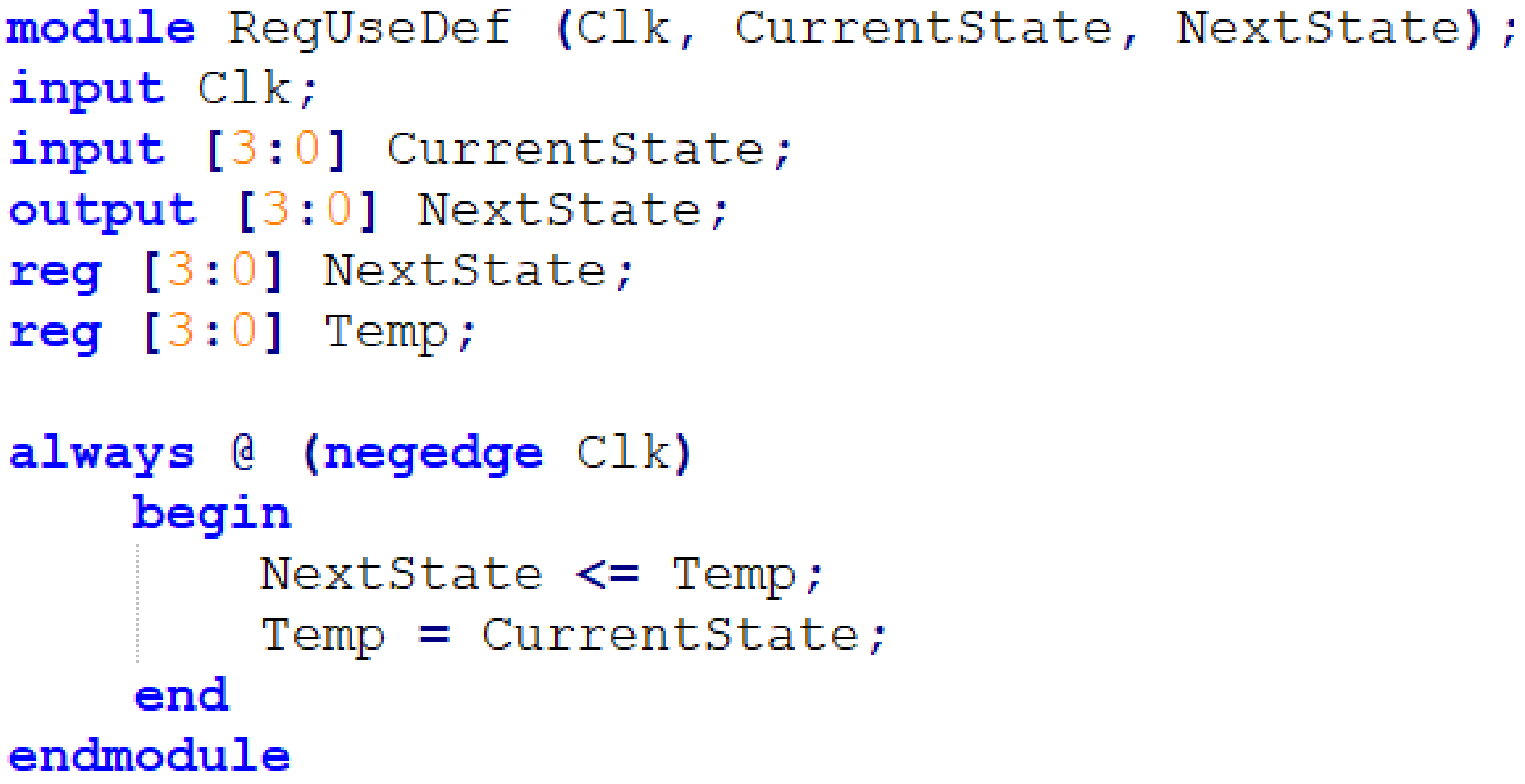
- What if a variable is defined globally (outside the always statement) but used only locally within an always statement?

```
module GlobalReg (Clk, CurrentState, NextState);
input Clk;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;
reg [3:0] Temp;

always @ (negedge Clk)
    begin
        Temp = CurrentState;
        NextState <= Temp;
    end
endmodule
```

Local Use of Variables

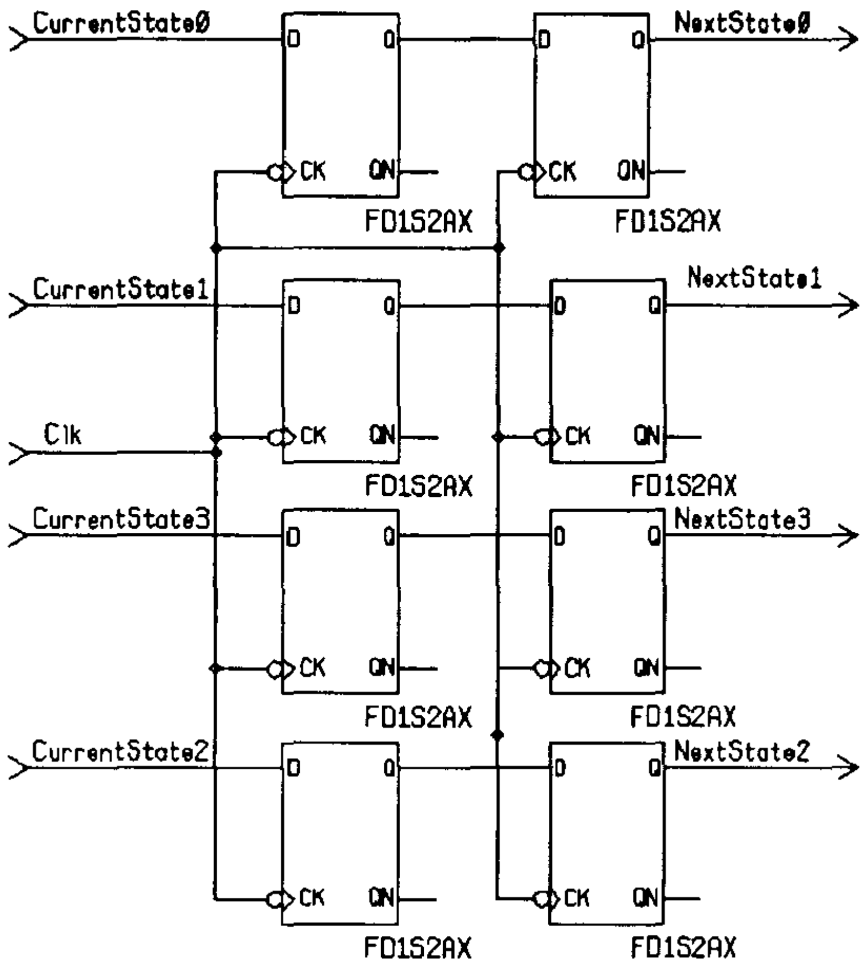


Local Use of Variables



```
module RegUseDef (Clk, CurrentState, NextState);
input Clk;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;
reg [3:0] Temp;

always @ (negedge Clk)
    begin
        NextState <= Temp;
        Temp = CurrentState;
    end
endmodule
```

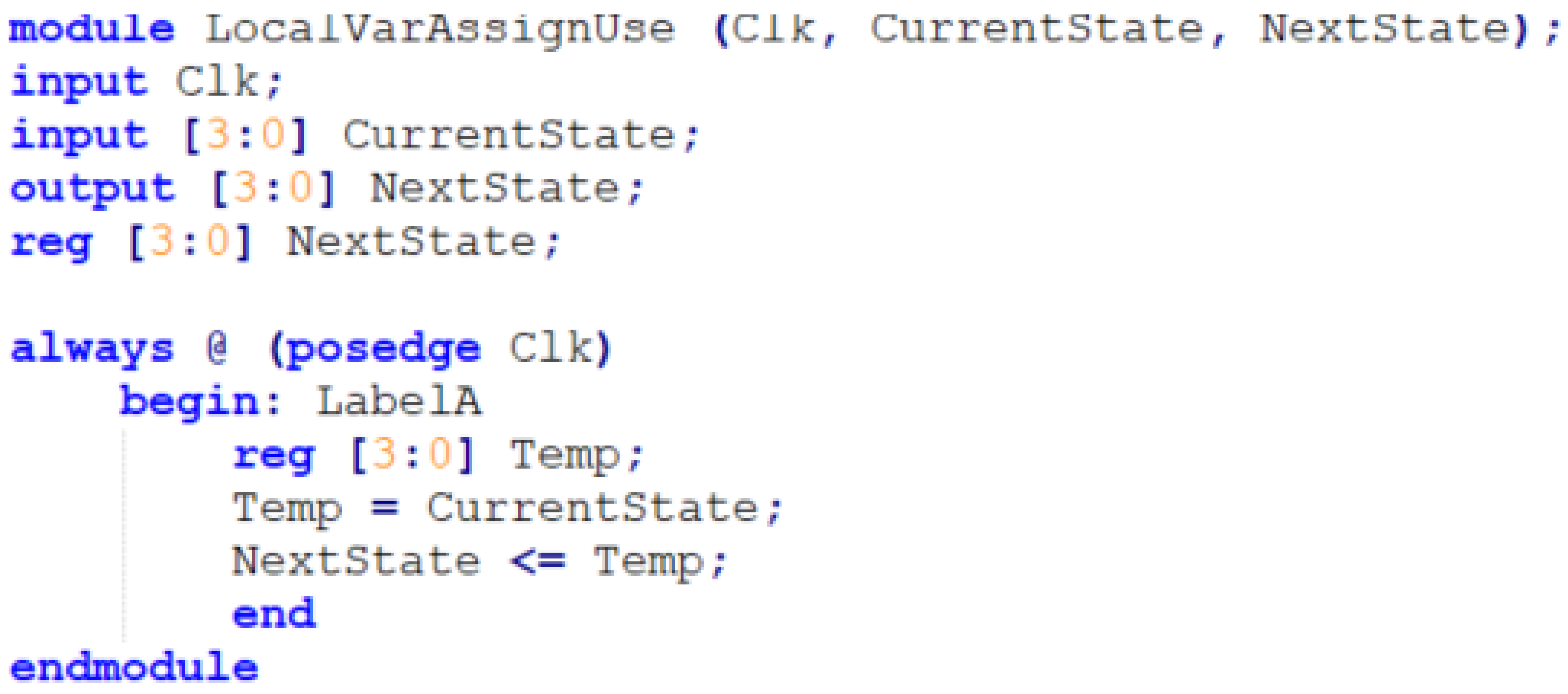
Local Use of Variables



A variable used before its definition is inferred as a flip-flop.

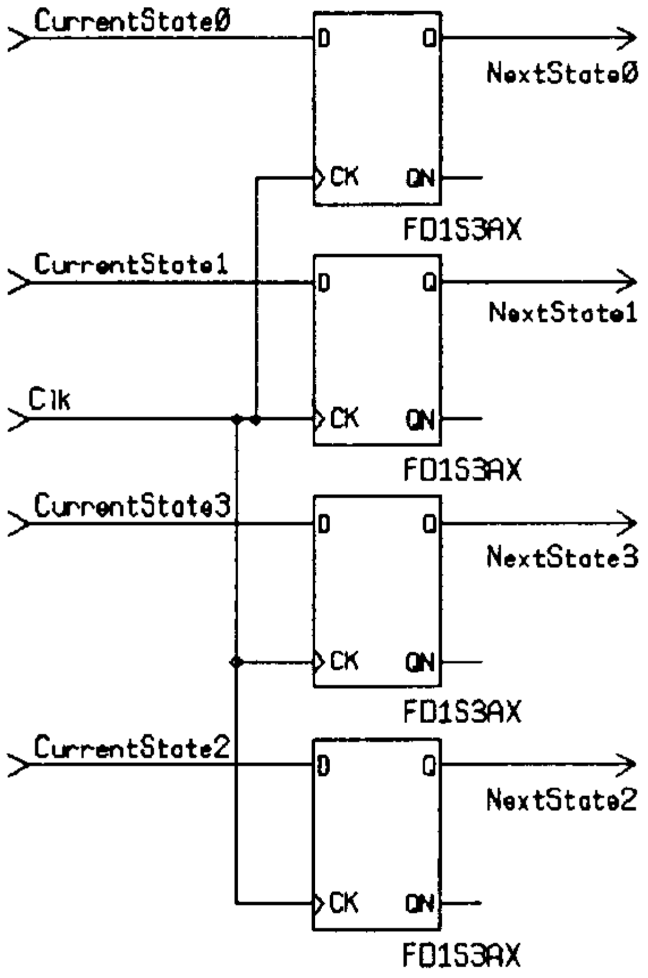
Local Use of Variables

- What if variables are declared locally within an always statement?



```
module LocalVarAssignUse (Clk, CurrentState, NextState);
input Clk;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;

always @ (posedge Clk)
    begin: LabelA
        reg [3:0] Temp;
        Temp = CurrentState;
        NextState <= Temp;
        end
endmodule
```

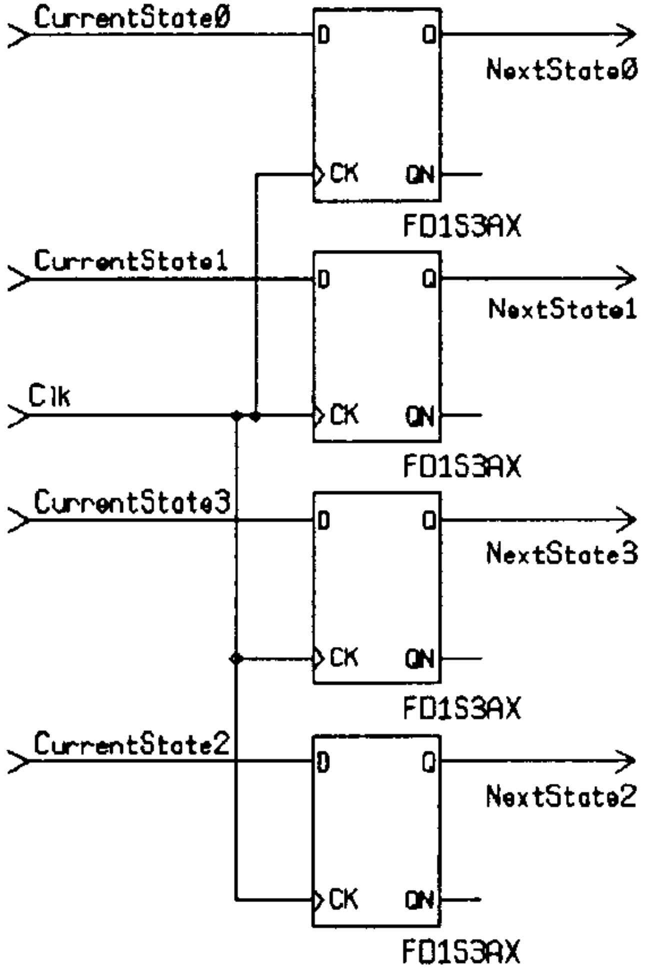
Local Use of Variables



Local Use of Variables

```
module LocalVarUseAssign (Clk, CurrentState, NextState);
input Clk;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;

always @ (posedge Clk)
    begin: LabelA
    reg [ 3:0] Temp;
        NextState <= Temp;
        Temp = CurrentState;
    end
endmodule
```

Local Use of Variables



Thank you