Digital Design with Verilog

Verilog - Synthesis

Lecture 21: Logic Synthesis with Verilog HDL – Part4

Objectives

- Inferring Latches in Case Statements
- Full Case / Parallel Case
- More on inferring Latches

Inferring Latches in case statements

- If a variable is assigned a value only in some branches of a case' statement, and not in all possible branches, then, a latch is inferred for that variable.

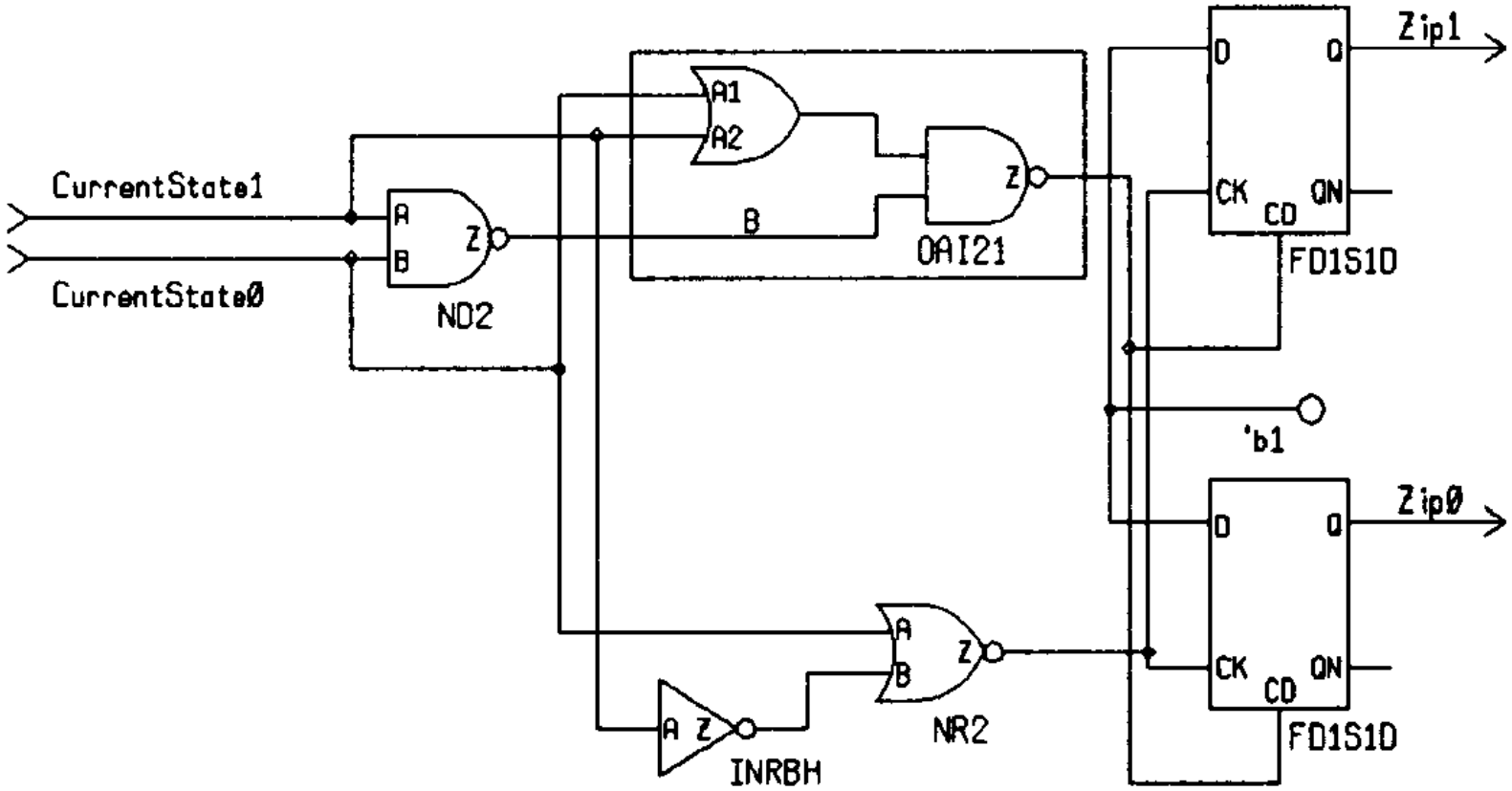
- The rules apply equally for casex and casez statements

Inferring Latches in case statements

```
module StateUpdate (CurrentState, Zip);
input [0:1] CurrentState;
output [0:1] Zip;
reg [0:1] Zip;
parameter S0= 0, S1 = 1, S2 = 2 , S3 = 3;
always @ (CurrentState)
     case (CurrentState)
          S0,
          S3: Zip= 0;
          S1: Zip= 3;
     endcase
endmodule
```

- In terms of latch inferencing, a case statement behaves identical to an if statement.

Inferring Latches in case statements

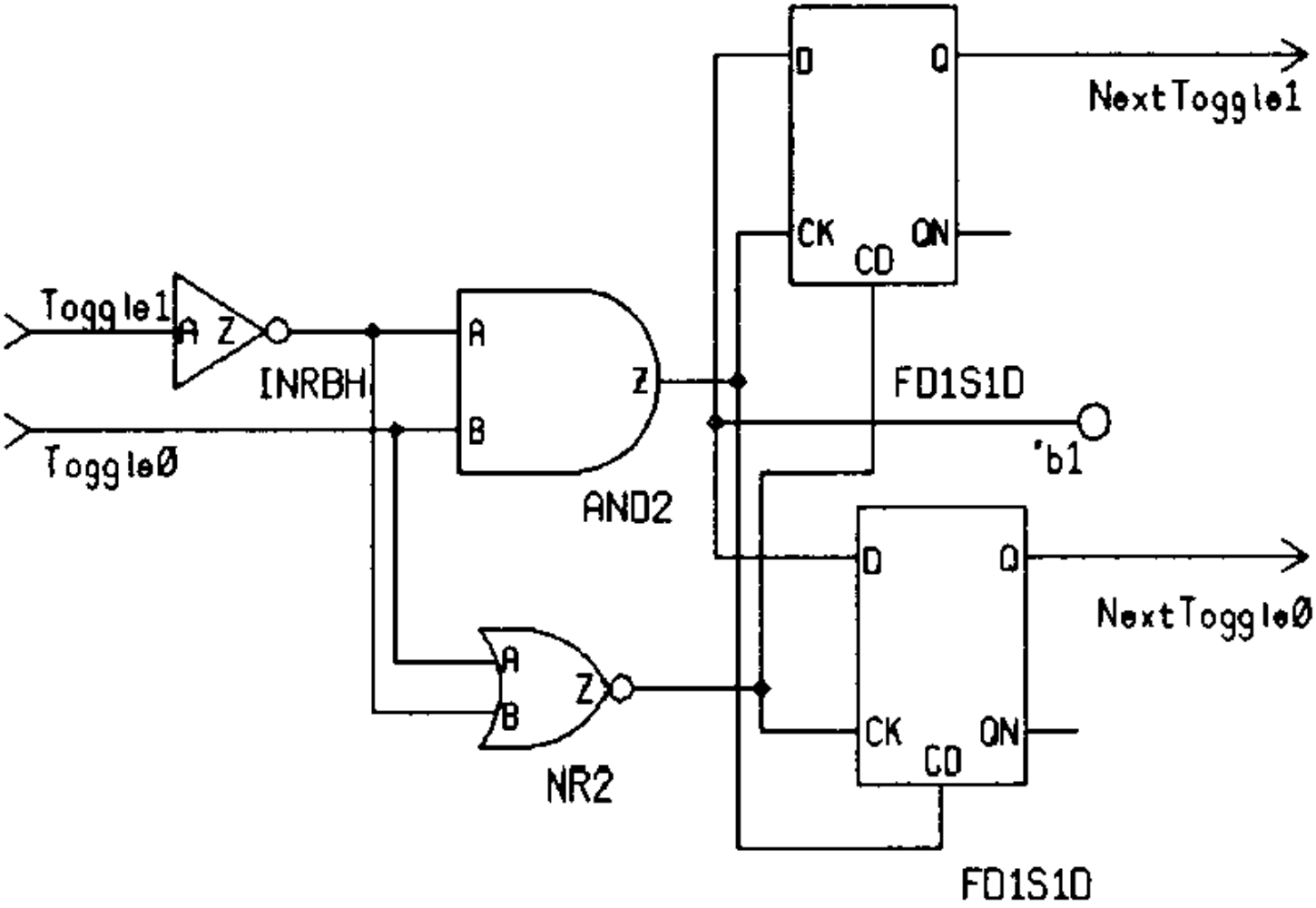


Latch inferred for a variable in a case statement.

Full Case

```
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
```

Full Case

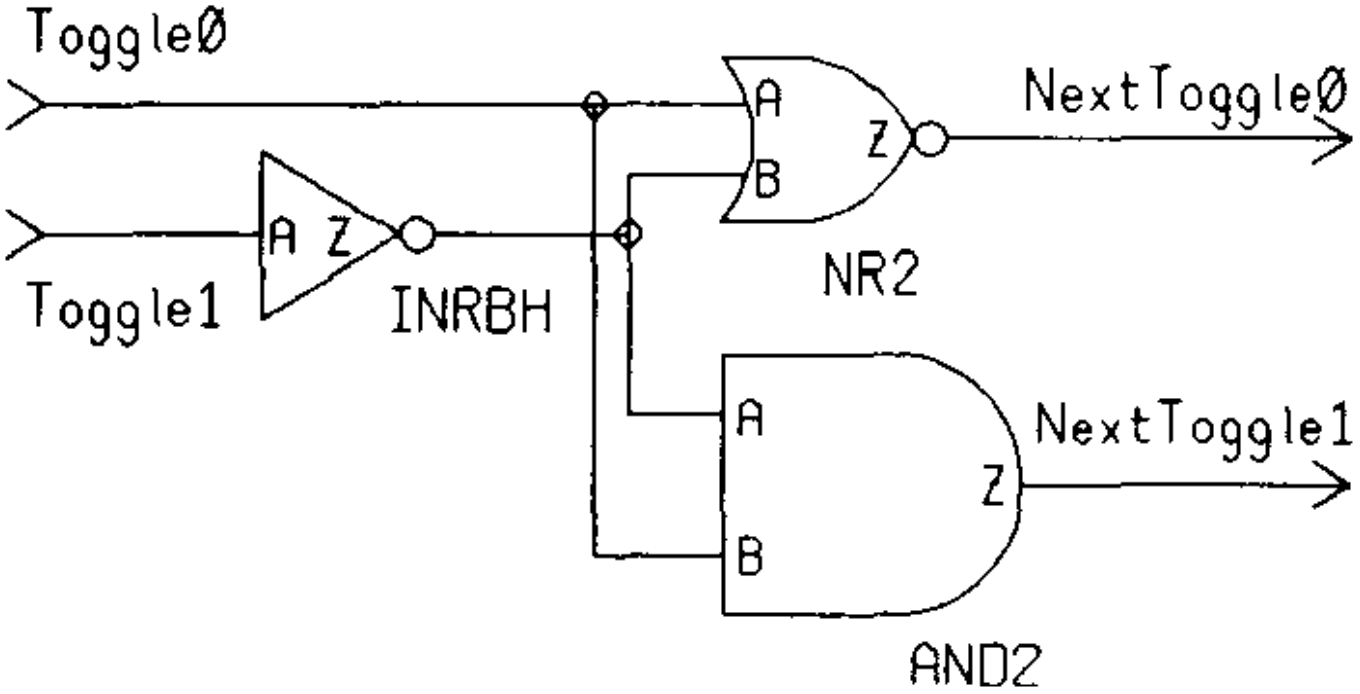


Latches are inferred for *NextToggle*.

Full Case

```
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
```

Full Case



With full_case synthesis directive: no latches.

Full Case (Alternate Style)

- To avoid latches, specify a default branch in the case statement or to make a default assignment to all variables assigned in a case statement.

```
always @ (Toggle)
    case (Toggle)
        2'b01 : NextToggle = 2'b10;
        2'b10 : NextToggle = 2'b01;
default : NextToggle =   2'b01;
//Dummy assignment.
endcase
```

```
always @ (Toggle)
begin
NextToggle = 2'b01; //Default assignment.
    case (Toggle)
        2'b01 : NextToggle = 2 'b10;
        2'b10: NextToggle = 2'b01;
    endcase
end
```

Parallel Case

- Verilog HDL semantics of a case statement specifies a priority order in which a case branch is selected.

- The case expression is checked with the first case item, if it is not the same, the next case item is checked, if not the same, the next case item is checked, and so on.

- A priority order of case item checking is implied by the case statement.

Parallel Case

- Verilog HDL, it is possible for two or more case item values to be the same or there may be overlapping case item values such as in casex and casez statements; however, because of the priority order, only the first one in the listed sequence of case items is selected.

- To apply the strict semantics of a case statement in synthesis to hardware, a nested if-like structure (priority logic: if this do this, else if this do this, else ...) is synthesized.

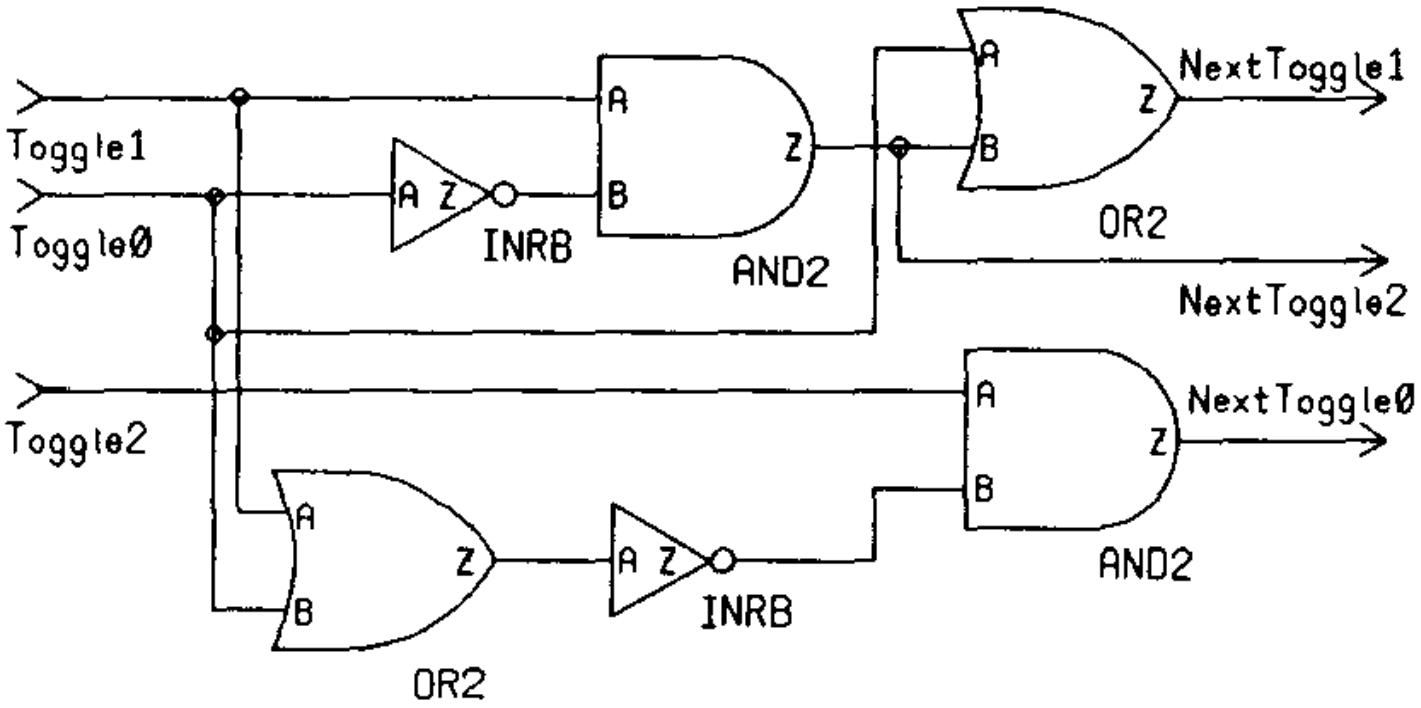
Parallel Case

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

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
```

Parallel Case

```
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;
```

Parallel Case



Priority logic selects each branch.

Parallel Case

- What if the designer knows that all case item values are mutually exclusive?

- In such a case, a decoder can be synthesized for a case statement control (the case expression is checked for all possible values of the case item values in parallel) instead of the priority logic (which could potentially be nested deep depending on the number of branches in the case statement).

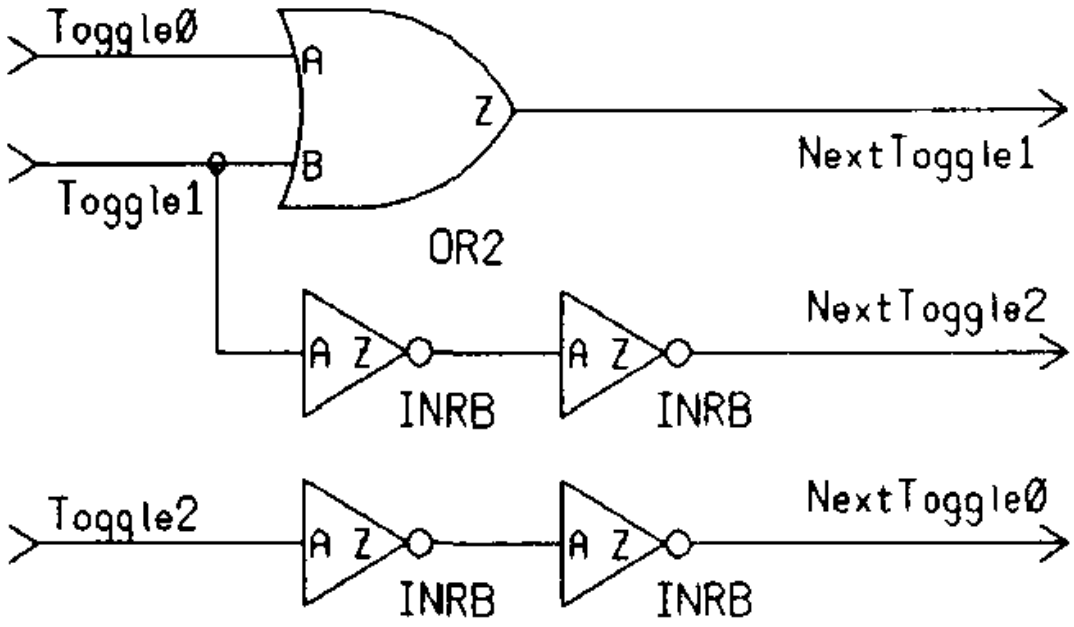
Parallel Case

- The information that all case item values are mutually exclusive needs to be passed to the synthesis tool.
- This is done by using a synthesis directive called *parallel_case.*
- When such a directive is attached to a case statement, a synthesis tool interprets the case statement as if all case items are mutually exclusive.

Parallel Case

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

always @ (Toggle)
    casex (Toggle) // synthesis parallel_case
        3'bxxl : NextToggle = 3'b010;
        3'bxlx : NextToggle = 3'b110;
        3'blxx : NextToggle = 3'b001;
        default: NextToggle = 3'b000;
endcase
endmodule
```

Parallel Case



With parallel_case directive: no priority logic.

Parallel Case

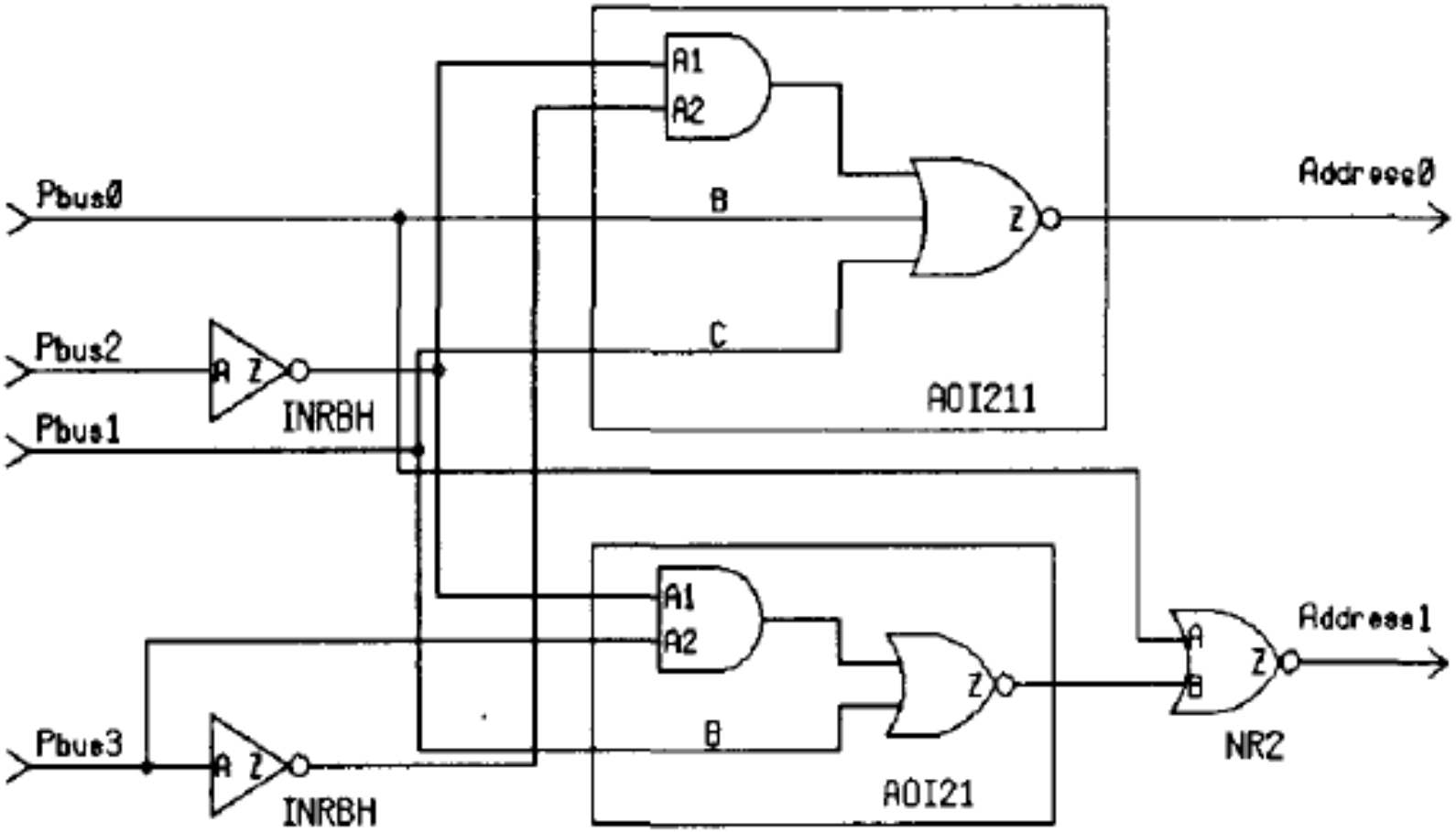
- The equivalent synthesis interpretation for the case statement is as follows (with only one if condition guaranteed to be true).

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

Non-constant as Case Item

```
module PriorityEncoder (Pbus, Address);
input [0:3] Pbus;
output [0:1] Address;
reg [0:1] Address;

    always @ (Pbus)
        case (1'b1)      ///synthesis full_case
            Pbus[0] : Address = 2'b00;
            Pbus[1] : Address = 2'b01;
            Pbus[2] : Address = 2'b10;
            Pbus[3] : Address = 2'b11;
        endcase
endmodule
```

Non-constant as Case Item



Priority encoder using case statement.

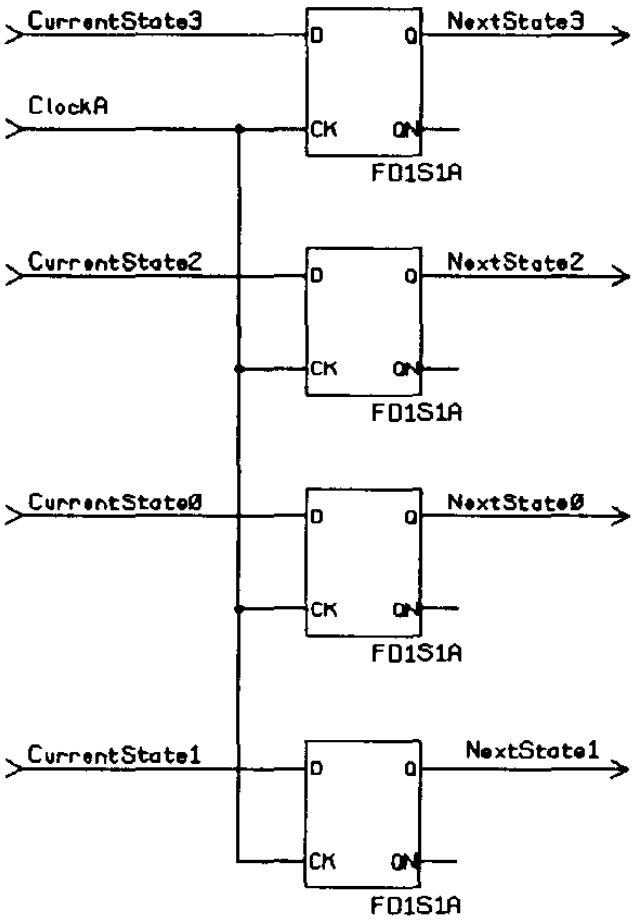
Non-constant as Case Item

```
always@(Pbus)
begin
    Address= 2'b00;
    case (1'bl)
        Pbus [0] : Address = 2'b00;
        Pbus[l] : Address= 2'b01;
        Pbus [2] : Address = 2'b10;
        Pbus[3] :Address= 2'bll;
    endcase
end
```

More on Inferring Latches

```
module LatchExample (ClockA, CurrentState, NextState);
input ClockA;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;
always @ (ClockA or CurrentState)
    if (ClockA)
        NextState = CurrentState;
endmodule
```

More on Inferring Latches



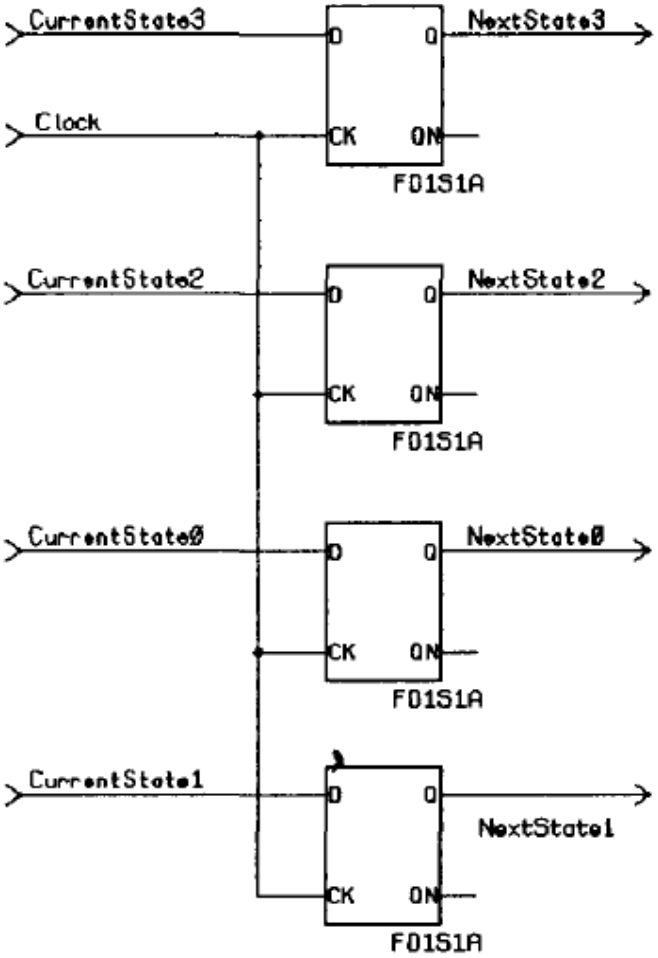
An incompletely specified condition infers a latch.

Locally Declared Variable

- A variable declared locally within an always statement is also inferred as a latch if it is incompletely assigned in a conditional statement (if statement or case statement).

```
module LocalintLatch (Clock, CurrentState, NextState);
input Clock;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;
always@(Clock or CurrentState)
    begin: Ll
    integer Temp;
    if (Clock)
        Temp = CurrentState;
    NextState = Temp
    end
endmodule
```

Locally Declared Variable



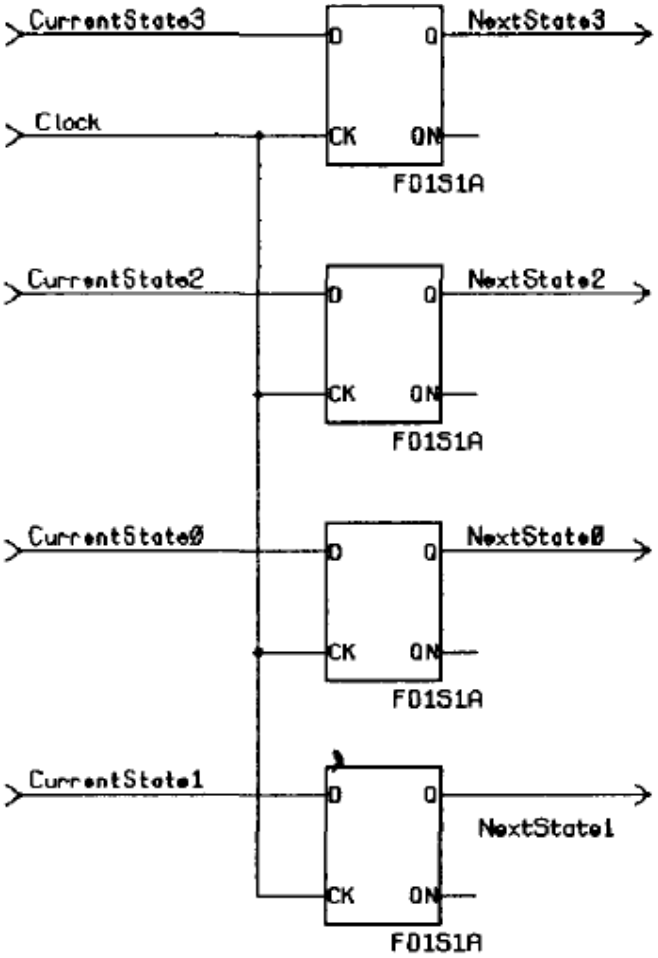
A local integer can also be a latch.

Variable Assigned Before Use

- If a variable is assigned and used within a conditional branch, no latch is necessary as shown in the following module. This is because the value of variable *Temp* need not be saved between level changes of *Clock.*

```
module LocalintNoLatch (Clock, CurrentState, NextState);
input Clock;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;
always @ (Clock or CurrentState)
    begin: Ll
    integer Temp;
        if (Clock)
            begin
                Temp = CurrentState;
                NextState = Temp;
            end
        end
endmodule
```

Variable Assigned Before Use

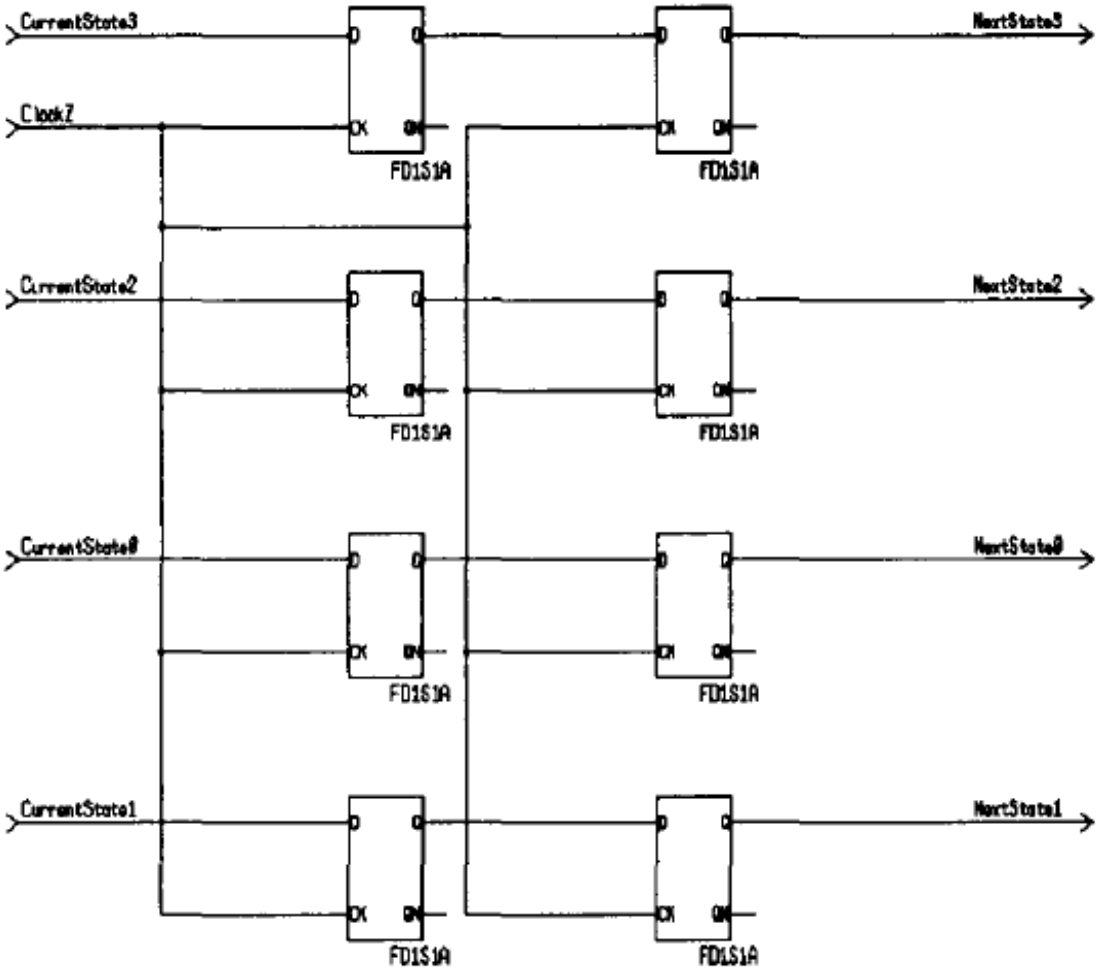


Use Before Assigned

- If a variable is used before it is assigned in an incompletely specified conditional statement, then a latch is inferred.

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

always @ (ClockZ or CurrentState or Temp)
    if (ClockZ)
    begin
        NextState = Temp;
        Temp = CurrentState;
    end
endmodule
```

Use Before Assigned



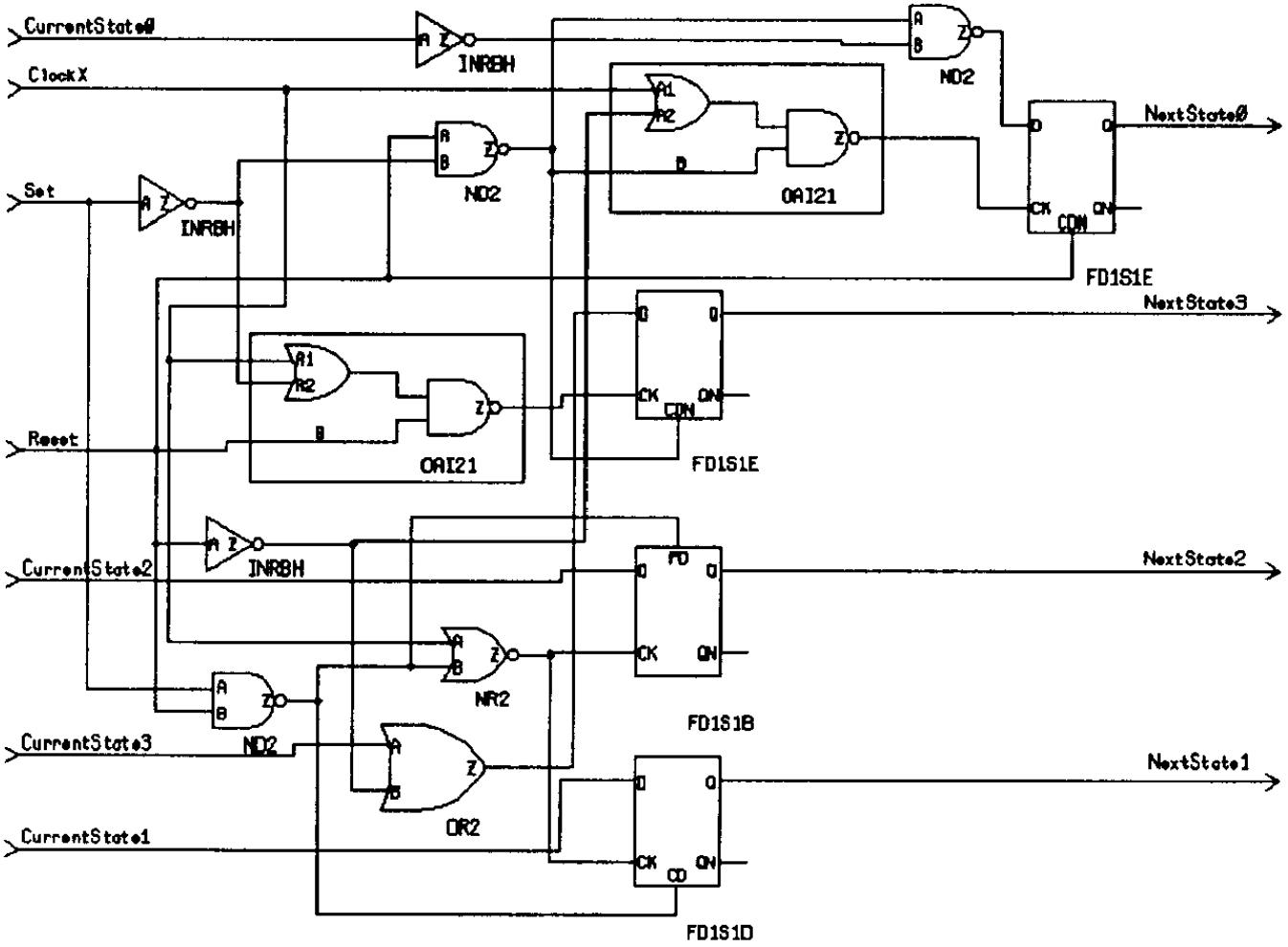
A variable used before being assigned in a conditional.

Latch with Asynchronous Preset and Clear

- If a variable, that is inferred as a latch, is assigned constant values in some branches of a conditional statement, bits that are 1 get assigned to the preset terminal of the latch, while those with 0 get assigned to the clear terminal

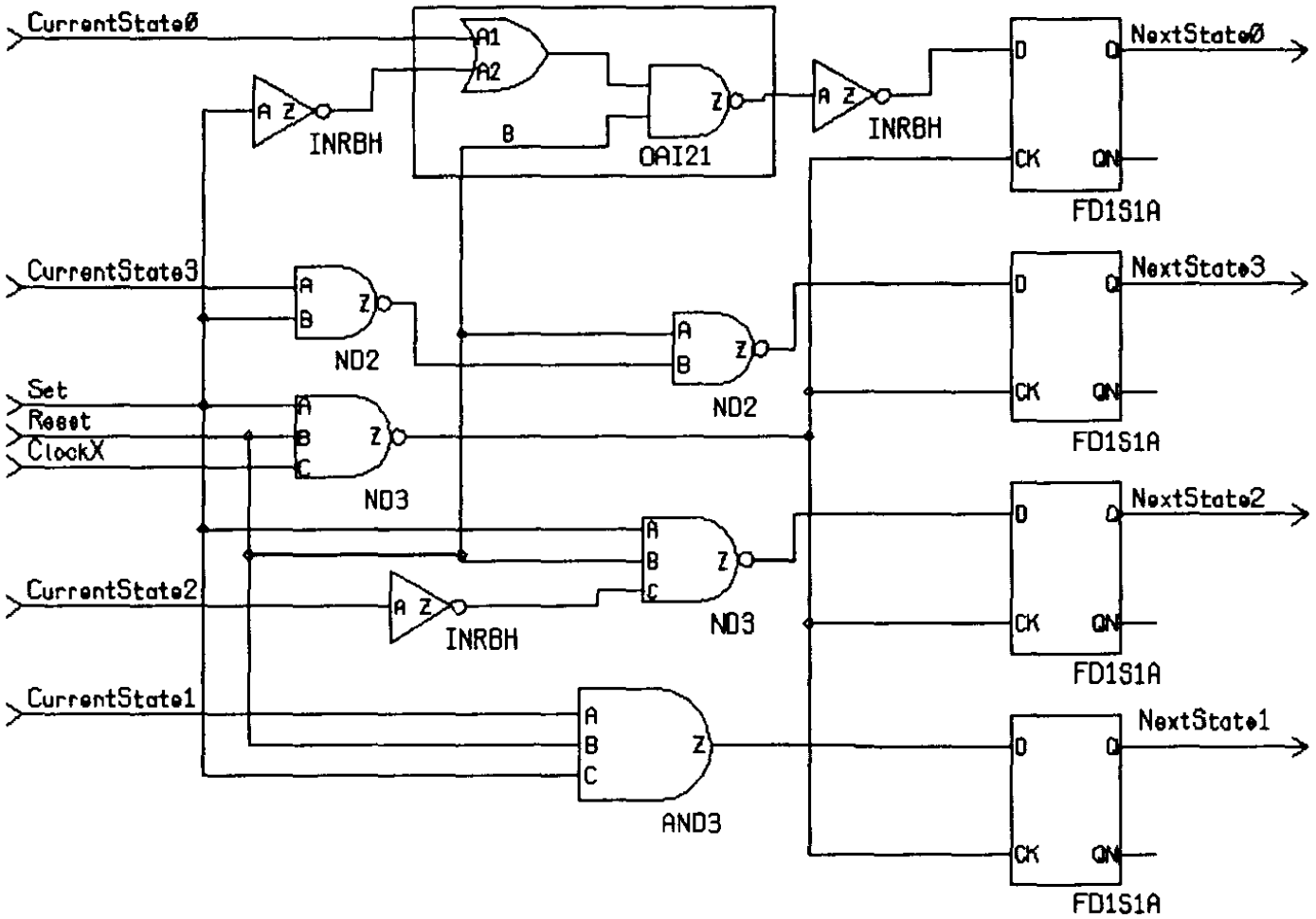
```
module AsyncLatch (ClockX, Reset, Set, CurrentState, NextState);
input ClockX, Reset, Set;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;

always @ (Reset or Set or ClockX or CurrentState)
    if (!Reset)
        NextState = 12;
    else if (!Set)
        NextState = 5;
    else if (! ClockX)
        NextState = CurrentState;
endmodule
```

Latch with Asynchronous Preset and Clear



Latch with asynchronous preset and clear.

Latch with Asynchronous Preset and Clear



Latches with no asynchronous preset and clear

Thank you